UNIVERSITY OF SOUTHAMPTON

An Adaptive Encoding of Geometric Shapes suitable for Aerodynamic Design using Genetic Algorithms

By

Robert Andrew Law

Thesis submitted for the degree of
Doctor of Philosophy

SCHOOL OF ENGINEERING SCIENCES, FLUID-STRUCTURE
INTERACTION RESEARCH GROUP

April 2002

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND APPLIED SCIENCES

SCHOOL OF ENGINEERING SCIENCES, FLUID-STRUCTURE INTERACTION RESEARCH GROUP

Doctor of Philosophy

An Adaptive Encoding of Geometric Shapes suitable for Aerodynamic Design using Genetic Algorithms

by Robert Andrew Law

Natural evolutionary systems exhibit a complex mapping between the genetic encoding carried by cells, to the body and form of a living species. The nature of this mapping facilitates the hereditary transfer of parental features to offspring through genes. Adaptation to this mapping occurs during the reproduction process, when parental chromosomes are blended together, and random mutations creep into this process.

Genetic Algorithms, which mimic evolutionary processes such as natural selection, reproduction, and survival of the fittest, can be applied to the problem of aerodynamic design, by breeding shapes together in the hope of finding better ones. Fixed chromosome structures are currently used to map the genetic encoding adapted by the Genetic Algorithm, to a geometric language that can be used to describe shapes such as airfoil sections or wings. To adequately encapsulate high quality aerodynamic shapes, large numbers of genes are required by this mapping at significant expense to the evolutionary process.

Suitable methods that reduce the computational time required to evolve aerodynamic shapes, may be sought by using an encoding that can add necessary detail to shapes, and adapting the complexity of its description.

In this thesis, the complexity and adaptation of shape encoding is explored. A distributed Genetic Algorithm has been created over clusters of networked PC's to perform aerodynamic optimisation. Different representations for describing shapes have been used to design airfoil sections. In order to reduce computational cost, metamodeling techniques were successfully implemented to predict which newly created shapes will be useful to the Genetic Algorithm, repairing breeding errors to increase design survivability. An object orientated chromosome framework has been developed, to facilitate adaptation of both genes and chromosome structure by Genetic Algorithms. A new hierarchical crossover operator is explored on evolving simple curves from straight lines, by adapting the complexity of the chromosome mapping used by Genetic Algorithm. Finally, the new adaptive encoding is exploited to evolve aerofoil sections, resulting in improvements to design quality and performance cost.

Contents

Abstract ii

Table of Contents iii

List of Tables ix

List of Figures x

Nomenclature xv

Glossary of Terminology Used xvii

Acknowledgments xx

Dedication xxi

1 Introduction 1

1.1 The Evolution of Shape 2

1.1.1 Evolving Aerodynamic Shape 3

1.1.2 Towards Shape Adaptivity 4

1.2 Aims and Objectives 5

1.2.1 Objectives 5

1.3 Layout of Thesis 7

2 Background 9

2.1 Design by Evolution 9
2.1.1 Adaptation and Natural Selection 10
2.1.2 Reproduction and Genetics 11
2.2 Artificial Evolution and The Genetic Algorithm 14
2.2.1 Evolutionary Algorithms 15
2.2.2 The Genetic Algorithm 16
2.2.3 Searching for the Optima 20
2.3 Approaches to Aerodynamic Shape Optimisation 23
2.3.1 The Application of GAs to CFD Optimisation 26
2.4 Representation and Parameterisation of Shape for Evolutionary Design 28
2.4.1 Airfoil Representation and Parameterisation in Shape Optimisation 28
2.4.2 Interpolation Functions 30
2.4.3 Three Dimensional Description of Aerodynamic Bodies 32
2.5 Evolving Geometric Complexity 34
2.6 Summary 36

3 Evaluation of Fluid dynamic Characteristics 38
3.1 Ducted Thruster Units 39
3.2 Characteristics of Propellers 40
3.3 Description of Aerodynamic Forces 41
3.3.1 Lift 42
3.3.2 Drag 44
3.3.3 Pitching Moment 44
3.4 Computational Analysis Techniques and the Panel Method 45
3.4.1 The Panel Method 46
3.4.2 Solution of Panel Characteristics 47
3.5 XFoil - Viscous Coupled Potential Solver 48
3.5.1 Laminar Boundary Layer Approximation 49
3.5.2 Implementation of the Turbulent Boundary Equations 51
3.5.3 Location of Transition 52

3.5.4 Implementation of Boundary Layer Calculation and Prediction of Drag . . . 53
3.5.5 Parameterisation of XFoil for Optimisation . . . 54
3.6 Unstructured Cell Vertex Euler Solver . . . 56
3.7 Summary . . . 57

4 The Formation of Species and Optima Finding with Genetic Algorithms 58

4.1 The Canonical Genetic Algorithms . . . 58
4.1.1 The Bump Problem . . . 59
4.1.2 GA initialisation, establishing generation zero . . . 60
4.1.3 Parental Selection . . . 64
4.1.4 Reproduction . . . 69
4.1.5 Death . . . 70
4.1.6 Implementation . . . 70
4.2 Performance Evaluation of the Canonical Genetic Algorithm . . . 71
4.3 Maintaining Population Diversity . . . 74
4.3.1 The Island Genetic Algorithm . . . 74
4.3.2 The Concept of Niche Formation . . . 77
4.3.3 Specification of Key GA Parameters . . . 81
4.3.4 Analysis of Speciation and Optima Finding . . . 83
4.4 Summary . . . 87

5 Distributing Expensive Objective Function Evaluations over an Office PC Network 88

5.1 Introduction to Parallel Genetic Algorithms . . . 88
5.2 Utilising Existing Computational Resources to Create a Commodity PC Network Suitable for Fast CFD Computation . . . 89
5.2.1 Creation of a Dual use computational facility . . . 90
5.2.2 Evaluation of the computational resource . . . 92
5.2.3 Cluster performance issues concerning dual boot implication . 102
5.3 Robust marine CFD on a dual-use distributed network . . . 103
5.3.1 Teaching Load on Network . . . 104

5.3.2 Failure Modes 105
5.3.3 Consequence of CFD Calculation Failure 105
5.3.4 Robust CFD Control Strategy 106
5.3.5 Control process 106
5.3.6 Control Implementation on an Euler Solver 107
5.3.7 Control Process for Genetic Algorithm Search with Panel Solver 110
5.4 Summary 111

6 Evaluation of airfoil shape parameterisation Using a Genetic Algorithm 112

6.1 Evaluating the Performance of Airfoil Sections Suitable for Ducted Thruster Units 113
6.1.1 The Cost Function 114
6.1.2 The Genetic Algorithm 115
6.1.3 Issues Concerning the Coupling of XFoil with the GA 116
6.2 Spline Representation Techniques 118
6.2.1 B-Spline Parameterisation 118
6.2.2 Bezier Representations 121
6.3 Analytical Functions 125
6.4 Efficient Parameterisation Through Ortho-Normalised Aero-Functions 132
6.4.1 Construction of the orthogonal set 134
6.4.2 Implementation of Ortho-Normalised Aero-Functions in Shape Optimisation 138
6.5 The Performance of Different Parameterisation Approaches 140
6.6 Sensitivity of a B-Spline Representation to Parameterisation 142
6.7 Summary 144

7 Adaption of Chromosome Complexity 146

7.1 Hierarchical Chromosome Framework 148
7.1.1 A Persistent Asynchronous Distributed Computing Environment via JINI-JavaSpaces 151
7.1.2 A Robust Farm-Worker Spaced Based Implementation 153
7.2 Hierarchical Crossover and Adaptive Mutation of a Line 154

7.2.1 Hierarchical Crossover 155
7.2.2 Hierarchical Mutation - Gene Addition and Deletion 159
7.2.3 Application of GA to evolve complexity of a simple curve . . . 161
7.3 Adapting the Complexity of a B-Spline curve 165
7.3.1 B-Spline Knot Insertion and Deletion 167
7.3.2 Implementation of Knot Insertion Within Complexity GA . . 168
7.3.3 Application of Adaptive Encoding on Aerodynamic Shape Optimisation 170
7.4 Summary 172

8 DNA Repair for Infeasible Offspring 175
8.1 The Effects of Representation on Lethal Crossover 176
8.2 Neural Network Classification of Infeasible Space 178
8.2.1 Feed Forward Neural Network 179
8.2.2 Classification using a Multi Layer Perception Network 179
8.2.3 Training the Network 180
8.2.4 Classification of Infeasible Airfoil Shapes 181
8.3 Meta-Modeling Using a Gaussian Process 182
8.3.1 Gaussian Processes 182
8.3.2 Gaussian Regression of the Bump Problem 183
8.3.3 Integration of Prediction Models with a Genetic Algorithm . . 184
8.4 Application of Airfoil Classification Service to GA Chromosome Repair 187
8.5 Summary 190

9 Conclusions and Recommendations 191
9.1 Conclusions 191
9.2 Further Conclusions and Observations 193
9.3 Recommendations 194

REFERENCES 198

APPENDICES 210

A Adaptive KMEANS Algorithm **211**

B Genetic Algorithm Implementation **212**

B.1 Genome Management Classes 212

B.2 Breeding Classes 216

B.3 Genetic Encoding Classes 217

C Normal Mode Analysis **218**

D Java Implementation of Adaptive Chromosome Encoding **220**

E Implementation of Gaussian Process **223**

List of Tables

2 A Sample of the Number of Chromosomes Found in Several Species . 12

3 Different Types of Mutation 14

4 Performance of Various Selection Schemes on 2D Bump problem (5 Samples) 72

5 Performance of Various Selection Schemes on 5D Bump problem (5 Samples) 72

6 Performance of Various Selection Schemes on 20D Bump problem(5 Samples) 72

7 Effect of Migration Rate and Interval on an iGA (6 demes of size 50) 77

8 Importance of Deme Size on an iGA ($mig_{int} = 20$, $mig_{rate} = 0.2$) . . 78

9 Communication Speedup Performance of Synchronous and Asynchronous GA on Modified Bump Problem 101

10 Airfoil Reconstruction Error from using Ortho-Normalised Aerofunctions 139

11 Resultant Best Drag Results Obtained from each Representation . . . 141

12 Coordinates of the line problems considered 162

13 Hierarchical Encoding Limits for Curve Points 163

14 Performance of Adaptive Line Tree Mutation when Compared with a Simple GA Implementation 164

15 Resultant Airfoil Performance Obtained Using Complexity BSpline Encoding 171

List of Figures

1 Schematic Illustration of the Crossover process 13

2 Schematic Diagram of the Main Processes in a Genetic Algorithm . . 17

3 Simplified Flow Diagram for a Simple Genetic Algorithm 18

4 Hypercube Representation of GA Search Space Encoding 19

5 Search Space Representation Featuring Several Local Optima's 22

6 Process for the calculation of cost sensitivities via the adjoint formulation 26

7 Mutation of a Primitive Type 35

8 Automated Design Process 35

9 Ducted Thruster Unit 40

10 Components of Forces acting on an Airfoil 42

11 Upper and Lower Surface Pressure Distributions for a NACA0012 Airfoil at $C_L = 0.6$ 43

12 Velocity potential at a point away from a vorticity distribution 47

13 XFoil Drag Polar Reconstruction for a NACA0012 Airfoil 55

14 XFoil Pressure Distribution Reconstruction for a NACA0012 Airfoil at $C_L = 0.0$ 55

15 2D Bump Problem 60

16 Stochastic Universal Sampling Scheme 68

17 Two Point Crossover Scheme 69

18 Implementation of the Canonical Genetic Algorithm 71

19 Typical Population Convergence (popsize 50, nGens 40) 73

20 Convergence of the Bump Problem with Tournament Selection (popSize 50) 74

21 Implementation of the Island Genetic Algorithm 75

22 Deme Topology and Migration Strategies 76
23 Immigration Control Algorithm for iGA (+1+2 Strategy) 77
24 Optimal Found with Deterministic Crowding (popSize 100, nGen 300) 81
25 Convergence of the Bump Problem with Niching (popSize 100) 82
26 Effect of Niching (popSize 100, nGen 300) 82
27 Search Trace of Best Members (popSize 100, nGen 300) 83
28 Rate of Species Creation for a Non-Niching GA 85
29 Rate of Species Creation for a Niching GA 86
30 Fitness Distribution of New Species vs Overall Best and Average Fitness 86

31 Schematic Diagram of the Departmental Teaching and Research Network 91
32 Distributed Upwind Algorithm 95
33 Semi-Scheduled Message Passing Algorithm 96
34 NACA 0012 Uniform Wing 97
35 Computational Domain for NACA 0012 Uniform Wing 98
36 NACA 0012 Uniform Wing Mesh 98
37 Mach Contours NACA 0012 $M_\infty = 0.85$ 99
38 Speedup performance of Euler Solver on PC Cluster 100
39 Parallel efficiency of Euler Solver on PC Cluster 100
40 XOR RAID Process 108
41 Distributed XOR RAID Communication Process between 8 Processors 109
42 XOR RAID Recovery Process 110
43 Speedup Performance for Euler Solver Using Regular RAID Tree Merge of Local Data 111

44 A Misleading Solution that Appears as a Good Solution 117
45 Poor Constraint Consideration May Lead to Unpractical Optima's . . 118
46 Limits used for B-spline representation with resulting section 120
47 Optimised b-spline section by perturbating ordinals about a reference section 120
48 Resultant best sections from implementations of full and delta parameterisations 121
49 Comparison of optimised pressure distributions 122

50 Resultant section when allowing BSpline poles to move in x and y directions . . . 122

51 Pressure distribution attained by allowing BSpline poles to move in x and y directions . . . 123

52 The resultant Optimised Section from a Bezier Spline Representation 124

53 Pressure distribution obtained using the Bezier representation ($C_L = 0.6$) 124

54 Comparison of optimised Bezier sections obtained using an iGA ($C_L = 0.6$) . . . 125

55 Superior convergence obtained through island Genetic Algorithm . . . 126

56 NACA Series Shape Functions . . . 127

57 Ortho-normalised modes of NACA Series Shape Functions . . . 128

58 Resultant sections obtained using orthogonal functions . . . 129

59 Optimal section of the orthogonal representation exceeds the delta b-spline boundaries . . . 129

60 Optimised pressure distribution from orthogonal representation . . . 130

61 CL = 0.3 Result with Tangential Geometric Condition Applied to Leading Edge . . . 131

62 CL = 0.6 Result with Tangential Geometric Condition Applied to Leading Edge . . . 131

63 Solution of a Foil Suitable for the Bi-Directional Operation of a TUUV Thruster Unit, $C_L = 0.3$ . . . 132

64 Solution of a Foil Suitable for the Bi-Directional Operation of a TUUV Thruster Unit, $C_L = 0.6$ . . . 133

65 Original Proposed Section formed through a rotated Kaplan K4-70 Section . . . 133

66 Drag Polar Comparison of Optimised Sections for Reversible Section . 134

67 Basis Airfoils for Constructing the Orthogonal Set . . . 135

68 Additional Basis Airfoils for Constructing the Orthogonal Set . . . 135

69 Basis Functions for Gram Schmidt Process . . . 136

70 Basis Functions from Additional Basis Airfoils . . . 136

71 Ortho-Normalised Airfoil Functions Modes 1-6 . . . 137

72 Ortho-Normalised Airfoil Functions Modes 7-10 . . . 138

73 Resultant Section Obtained using Ortho-Normalised Aerofunctions . . 140

74 Effect of adding more control points to B-Spline parameterisation . . 143

75 Hierarchical relationship between Form, Surface, Curve and Point entities 148

76 Adapting a line into a curve using a B-Spline representation, under the Hierarchical Framework this adaption can use two mutations, one to add the extra genes (new Point), the other to mutate the new genes . 149

77 Hierarchical Implementation of a Simple Curve Chromosome 150

78 Adaption of Hierarchical Curve Chromosome by Adding New Genes in the Form of an Extra Point 150

79 New shapes can easily be created once new points have been added . 151

80 Simple JavaSpace Model of Farmer-Worker 153

81 JavaSpace Farm Implementation within GA Evaluate 155

82 Algorithm for JavaSpace Worker 156

83 Chromosome Structure for a Set of Connected Lines 156

84 Parent gene exchange via sub-tree swapping 157

85 Hierarchical crossover at sub-level 157

86 Hierarchical Crossover Algorithm 158

87 Point Crossover Implementation 158

88 Algorithm for Hierarchical Crossover about Common Gene Points . . 159

89 The effect of sub-tree deletion on a simple curve 161

90 Line Curve Point Insertion Algorithm 161

91 The effect of sub-tree addition on the curve and defining chromosome structure 162

92 Evolution of fitness 164

93 Number of unique individuals in each population 165

94 Number of similar genes shared between breeding parents 166

95 Crossover efficiency 166

96 Reconstruction of NACA0012 using GA with Fixed Encoding 168

97 Reconstruction of NACA0012 using GA with Adaptive Encoding . . . 169

98 NACA0012 Reconstruction Convergence for Fixed Gene Encoding . . 170

99 Reconstruction of NACA0012 using Powell's Direction Set Method . . 171

100 Airfoil Optimisation Convergence Using Complexity BSpline 172

101 Number of Genes Used per Generation in the Evolution of $C_L = 0.6$ Complexity Airfoil 173

102 Resultant Airfoil Section from Using Complexity BSpline Representation 173

103 Resultant Airfoil Pressure Distribution Obtained Using Complexity BSpline Representation . . . 173

104 Resultant drag reduction from perturbation method, and the effect of increasing the search space on infeasible solutions . . . 177

105 Effect of different representation methods on infeasible solutions . . . 177

106 Objective surface for using first two orthogonal modes of the orthonormalised Aerofunctions at $C_L = 0.6$ . . . 178

107 Infeasible Search Space (no solution obtained from Xfoil) . . . 181

108 Neural Network Prediction of Infeasible Space . . . 182

109 The Bump Function . . . 184

110 Gaussian Process Metamodel of Bump Function . . . 185

111 The Absolute Difference Between Bump Metamodel and Bump Function 185

112 Distributed Genetic Repair and Identification Algorithm . . . 187

113 Distributed Airfoil Classification Service . . . 188

114 Airfoil Regression . . . 189

115 The effect of airfoil repeir . . . 189

116 A Simplified UML diagram showing hierarchical relationships of the main classes that implement the GA . . . 213

117 Population reinsertion scheme implementation for Crowding control . 213

118 Genome fitness scaling implementation UML . . . 214

119 Parent selection scheme implementation UML . . . 214

120 UML diagram of fitness evaluation via a penalty scheme . . . 215

121 UML Diagram of the Hierarchical Chromosome Encoding Framework Used to Adapt Encoding Complexity . . . 221

122 UML Diagram of the Complexity Based B-Spline Encoding Implementation . . . 221

123 Airfoil Encoding Using Complexity Based B-Splines for the Upper and Lower Curves . . . 222

NOMENCLATURE

c	foil chord length
C_D	drag coefficient
C_f	skin friction coefficient
C_L	lift coefficient
C_M	moment coefficient
C_P	pressure coefficient
D	drag force
f^*	numerical flux
$\vec{F}$	flux vector with components f, g, h
H	boundary layer shape factor
L	lift force
M	pitching moment
$\vec{n}$	normal vector
p	pressure
q	dynamic pressure
Q	torque
Re	reynolds number
S	control volume face area
t	time
T	thrust
u	scalar quantity
U	state vector
U_∞	Free Stream Velocity
$\vec{v}$	velocity with Cartesian components u, v, w
$\acute{v}$	disturbed velocity
V	vector of primitive variables, volume, velocity
x, y, z	cartesian coordinates
α	angle of attack
δ^*	boundary layer displacement thickness
Ω	control volume
μ	doublet strength

ϕ	velocity potential
Φ	total velocity potential
ρ	density
σ	source strength
θ	angle, boundary layer momentum thickness

GLOSSARY OF TERMINOLOGY USED

Adaptation: The general advance in a members ability to survive within a changing environment.

Allele: An element found at a specific position (locus) on the chromosome.

Chromosome: Rod-shaped bodies in the nucleus of a cell, which contain the genes.

Classifier System: Dynamic, rule based systems that are capable of learning and classifying information sets.

Crossover: A process by which genetic information is exchanged between chromosomes.

Deme: An independent population of evolving members based on a migration model.

DNA: Deoxyribonucleic acid, a double stranded helix structure of base pairs that determine the makeup of protein during embryology.

Elitism: A process by which the most fit members from one generation are copied to the next.

Fitness: A user defined measure that is used by the Genetic Algorithm to determine how well a candidate performs against the objective function.

Gene: An encoding for the synthesis for a protein that can be passed from parent to offspring.

Genetic Algorithm: An evolutionary algorithm based on the mechanics of natural selection and survival of the fittest.

Genetic Programming: An evolving computer algorithm based on the principles of the Genetic Algorithm.

Genome: A member of an evolving population that contains the total genetic information of the organism.

Genotype: The genetic construction of the design variables contained within the chromosome.

Gradient Method: Local optimisation algorithms based on the first partial derivatives of the objective function.

Hamming Distance: For two binary vectors, the Hamming distance is the number of different positions.

Infeasible: A solution that cannot be fully obtained through the fitness function.

JavaSpaces: A persistent distributed Java implementation for the exchange of objects between Java processes.

Meta-Modeling: A pseudo model that attempts to represent real information or models.

Migration: The transfer of an individual from one sub-population to another.

Natural Selection: The result of competitive exclusion as organisms compete to fill a finite resource space.

Neural Network: An artificial implementation of the data processing capabilities of brains on a computer.

Niche: The set of possible environments that permit survival of a species.

Objective Function: A user defined function used by the GA to determine the fitness of a candidate.

Penalty Function: A constraint handling function that depreciates objective performance depending on the violation of constraints.

Parameterisation: The definition used to determine the range and length of the design variables, and hence the size of a members chromosome.

Phenotype: Resultant features of a members genetic makeup such as colour, height and shape.

Population: A group of individuals that may interact with each other within an evolutionary process.

Representation: The process of constructing the object whose fitness is to be measured from its chromosome.

RMI: Remote Method Invocation, a distributed Java server implementation.

Schema: A schema describes a subset of all binary vectors of fixed length that have similarities at certain positions.

Species: A population of similarly constructed organisms, capable of producing fertile offspring. Members of one species occupy the same ecological niche.

ACKNOWLEDGEMENTS

I would like to thank the staff and students of the Fluid-Structure Interaction Research Group for their help and friendship throughout this work, and Dr. Denis Nicole for his advise and assistance in creating the distributed computing facility. I would particulary like to thank Dr. Steve Turnock for his boundless support and enthusiasm for my work, especially through difficult times, and various extended escapades across the world.

I would especially like to give thanks to Carren Holden of BAESYSTEMS who introduced me to this wonderful field of research. She has become a great friend since we met at Airbus, and who's inspiration and the countless enthusiastic discussions on this subject, has helped to guide and shape the direction of my work.

Mere thanks is not enough acknowledgement of how much support my parents have given me over the past four years. Nor to my friend and best man Alistair, to whom I am grossly in debt (literally), for the number of drinks he has bought me to support the burden of postgraduate life.

Finally, to the patience and love of my wife Louise, who's countless support I have depended on so much, it is to her I dedicate this thesis.

This work has been supported by BAESYSTEMS, and an EPSRC grant.

To Louise

Chapter 1

Introduction

The world we live in exhibits complexity and richness beyond our level of comprehension. We are surrounded by complex objects, whether they are biological such as ants, birds, fungi, elephants, or human made like a car, a telescope, or an aircraft. What differentiates these objects from other simple objects such as rocks, planets, and clouds is that in some way their design appears to have evolved around purpose.

It is not that a rock, cloud or planet cannot have a purpose, indeed rocks are used to provide terrain or shelter, clouds provide weather systems, and planets provide habitation and resources that are a necessity for life. What differentiates these objects from others like cars, jellyfish, birds etc., is that their design does not appear to have been influenced by any useful purpose such as transportation or survival. Instead, they are just clouds of tiny particles, lumps of uniform matter like crystals, or piles of matter haphazardly arranged into the planet, mountains, and clouds that we see.

A car, however, is designed specifically to transport people and objects. It consists of many other specifically designed components like a body, four wheels, a crankshaft, pistons, and a combustion chamber. What is more interesting about these components is not just that they too have been purposely designed, but that they all interact in unique ways with other components. Therefore a car is a collection of objects that purposely interact with one another to satisfy their combined purpose.

On the other extreme, the purpose that a bird serves is less transparent. If one accepts the Darwinian argument that the purpose of a species is to survive and pass on its genes to further generations through breeding, then we can easily accept that the purpose of the birds wings are not deliberate, but a consequence that was able to help the bird in its greater purpose. The fact that wings can make a bird fly is amazing by itself, and even more so when considering their design. Feathers which are particular to that species, have become a common component in the wing construction. The design of the feather changes slightly depending on their location on the wing and species of bird. Yet they work seamlessly together with the wing bones, muscle, flesh and other feathers to provide a bird with the means to fly with perfect aerodynamics and control when compared to the latest human flying contraptions.

If complex objects are merely a consequence of purpose, then the components of an aircraft do not by themselves justify a complex system capable of flying, any different to a mountain being a collection of objects. To actually fulfil the purpose, the aircraft components must be put together in one of only several out of millions of arrangements This is the miracle of evolution. So many things can go wrong!

1.1 The Evolution of Shape

Natural evolution is a system found in nature that many naturalists believe to have had great bearing on the evolution of prehistoric living organisms, into the natural life that we know today. Although we have only a basic insight into the actual mechanics of such a complicated system, evolutionary mechanisms borrowed from nature can be employed in artificial evolutionary processes.

Evolutionary Algorithms (EA) [1] attempt to mirror the fundamental processes of natural evolution, and have been applied to a variety of design search problems. Such algorithms use processes such as natural selection, and reproduction to ensure that productive traits are propagated into future generations. Movement in the direction of improvement, and successful combinations of design properties from several members are ensured through artificial analogies to natural selection, survival of the fittest, and sexual reproduction. Although evolutionary algorithms can outperform traditional optimisation approaches when applied to problems such as aerodynamic design, they have yet to produce any design that exhibits the characteristics of complexity dominant in the man made objects that have changed our way of living today.

1.1.1 Evolving Aerodynamic Shape

In aerodynamic design, a slight change in the shape of a body may have complex and even chaotic non-linear effects on the resulting flow system. For example, in the design of an airfoil, aerodynamicists may push the trailing edge surface for improved drag performance, at the risk of introducing flow separation at other speeds of operation. Such responses can easily cause search processes to become trapped where neither search direction can offer any improvement over the existing design. Such a design point is known as a local optimum. No search algorithm is immune to such traps, although one type of evolutionary algorithm, the Genetic Algorithm (GA) has proved to be significantly more robust in searching such an environment than traditional approaches. This ability has allowed GAs to find more novel and significantly better designs than other search processes when applied to aerodynamic shape optimisation problems.

The main restriction to the use of Genetic Algorithms in shape optimisation is the associated cost of evaluating each design candidate. A means of easing this cost has focused research efforts in two areas: more efficient algorithms that can find good solutions with fewer design evaluations, and more efficient shape descriptions that require fewer design variables to be specified by search processes. Both of these quests carry a risk of deteriorating the ability of the Genetic Algorithm to find the optima.

One important motivation of evolutionary design that has been almost forgotten by the aerodynamic design community is the synthesis of adaptation. The introduction of such an important feature in shape design, could allow the iterative adaption of shape to arbitrary complexity. The benefits to the design process that this feature could offer include: the creation of complex shapes from simple idea's or components; the ability to traverse larger search spaces; a decoupling of the search algorithm's relationship with the initial design, and its definition; the reuse of solutions; solution adaptation to changes in the objective environment; the potential to search complex spaces more efficiently.

One of the main bottlenecks in shape adaptivity in traditional applications of GAs for shape optimisation, is the fixed pre-defined coupling between the language and words used to define a shape, and its genetic implementation used in GAs to apply breeding and mutation operations. The description of curves and surfaces is well known, and a variety of methods are available to ensure that accurate geometric representations are well defined. This rich knowledge base provides a suitable platform from which to explore the adaptability of shapes using GAs in a shape optimisation environment.

1.1.2 Towards Shape Adaptivity

To write down the form of a circle, one may choose to describe its properties in terms of the coordinates of its centre, and the length of its radius. For a very different shape such as a square, the coordinates of each vertex may be used. However neither description would be sufficient to describe a square with arced sides.

In order to describe all these objects, we may need to adopt a more complex language of description. One such language may use four arcs, describing a circle as a collection of arcs of the same curvature, and a square as a collection of arcs of infinite curvature. Now, this language can adequately include our box with arced edges, as well as a whole collection of other four sided shapes. However, to go on describing even more complex shapes such as hexagons and stars, we would need to add more arcs to this language.

Two and three dimensional shapes can be constructed from a variety of descriptions including primitive forms such as arcs, circles and lines, to interpolated curves such as those used in Computer Aided Design (CAD). However, when such shape representations are used for the purpose of adapting a shape in an evolutionary cycle, thought must also be given to the manner in which these shapes can be mutated and bred together.

To evolve shape within a computer assisted evolutionary system, the following operations are required.

- A language for describing shapes.

- The ability to mutate shapes.
- A means of breeding shapes together.
- A measure to evaluate the performance of one shape relative to another.

This work focuses on the first three of these requirements, paying close interest to the range and geometric sensitivity of the shape description scheme used when applied to a simple airfoil design problem. An airfoil may appear to be a relatively simple shape, but in the presence of a flowfield, its shape becomes the critical component of a complex non-linear system. A slight shift in position or change in shape will effect the entire flow system surrounding the body, from affecting the forces acting on the body, to creating chaotic turbulent flows.

The particular question under investigation in this work, is how to improve the adaptiveness and complexity of an evolving shape. Through such an improvement, a potentially larger search space can be explored, without significantly increasing the number of designs that need to be evaluated during the evolution process. Complexity, shape refinement and sensitivity can be sought, that would not have been possible without using a large number of design parameters in previous implementations.

Previous research [2][3][4][5], has tended to focus on how a shape should be described in a GA process for efficiency reasons. This work is set out to examine how a description for shape can best support adaptation.

1.2 Aims and Objectives

The aim of this work is an improved language for describing the form of shapes used in an evolutionary based aerodynamic design process. Both the language and its complexity should be allowed to adapt in order to define high quality aerodynamic shapes.

1.2.1 Objectives

In order to attain this aim, the objectives of this work are to:

- understand the performance and limitations of a Genetic Algorithm in evolving simple language descriptions;
- establish a cheap distributed computational platform for evaluating aerodynamic designs sampled during an optimisation process;
- produce and optimised aerofoil and investigate the effectiveness of different geometric representation techniques in supporting the evolution of aerodynamic shapes;
- develop a framework for adapting the complexity of a language, and demonstrate the ability of a Genetic Algorithm to use this framework to evolve complexity;
- evolve shape complexity using a Genetic Algorithm and incorporate this facility into an aerodynamic optimisation process;
- explore new opportunities to reduce the computational cost of aerodynamic evolution, without depreciating search performance.

1.3 Layout of Thesis

The application, background, and motivation for using Genetic Algorithms with aerodynamic design methods is introduced in the next chapter, with reference to concerns and observations made throughout the research community into the quality of shape that can be evolved by such a system. Methods for representing curves for airfoil description are reviewed and possible approaches and limitations towards improving shape adaption explored.

Chapter three presents the development of a Genetic Algorithm for searching difficult optimisation problems. Focus is given to the problem of population diversity maintenance to ensure tolerance towards local optima traps. The convergence and dynamics of evolving populations are explored to encourage adaption and robustness, through the application of a variety of speciation schemes.

In chapter four the performance of a distributed computational platform built from a network of Office PC's is discussed. The speedup performance and platform stability of the network is evaluated on a parallel CFD problem. An efficient asynchronous communication algorithm has been developed for the GA to make effective use of the network for aerodynamic shape optimisation. To address problems concerning PC network reliability and robustness within an office environment, a scalable backup facility has been developed to allow the quick backup of distributed data using a process motivated by disk raid storage.

A study into the use of several geometric modeling techniques with Genetic Algorithms is given in chapter five. The ability for each representation to obtain smooth airfoil shapes is investigated, following application of a GA to several aerodynamic design problems. A new representation technique based on ortho-normalised airfoil function analysis has been developed and tested on an airfoil design problem using the GA. A novel design problem seeking a reversible airfoil whose performance characteristics are equal in either flow direction, was also explored using the GA.

Chapter six investigates the problem of adapting the language used to describe shape, to improve the design range and adaptiveness of the evolution process on shape design problems. A framework is proposed to facilitate this adaption, and includes several object based genetic operators to assist in breeding different shapes together. A hierarchical chromosome framework is implemented for the encoding of curves that ensures the principle of gradual evolution. The new chromosome is allowed to adapt its genetic complexity by adding and deleting new genes. Several curve reconstruction problems are explored using the new encoding, starting from a simple line defined by just two points. Finally, the adaptive encoding is applied to aerodynamic shape optimisation based on a B-Spline representation.

To approach the problem of infeasible shapes produced by the crossover process, that can lead to non-convergence in the CFD analysis, a form of genetic repair is explored in Chapter seven. A meta-modeling process is proposed to predict aerodynamic analyse failure, so that wasteful computation in obtaining fitness results for airfoil shapes can be avoided. A Multi-Layer Perception model and a Gaussian Process was applied to classification and regression problems involving fitness landscapes. The gaussian process was also applied to predicting evaluation failures in aerodynamic shape optimisation.

Chapter eight concludes the research presented in this work, and draws on observations made from this research, and recommends new directions for future research.

Chapter 2

Background

This work is concerned with the idea of "Design by Evolution", and in particular on the properties, characteristics and environment necessary to form highly adapted aerodynamic shapes. In this chapter, a foundation into evolutionary shape optimisation is given, and issues affecting the adaptation of shape discussed, defining the key areas of research detailed by this thesis.

2.1 Design by Evolution

Establishing the behaviour and characteristics that determine whether a system is complex or not, is a subject of much debate. Dawkins [6] indicated that 'design for purpose' would provide a good guideline towards this question. It is easy to understand that an eye may have evolved from the purpose of enabling animals to survive better assisted by sight, whereas it seems more difficult to accept that the precise layout of rocks evolved from a need for mountains and hills. However, design for purpose alone is not enough to define whether an object is complex or not. For example, would one call an object such as a hammer complex? The hammer is a well-evolved design, which originally introduced its function to mankind as a rock. Today, hammers are well balanced, consist of a heavy head and a light handle designed to pass maximum kinetic energy from the hammer to the object being struck. Dawkins went onto refine his definition by suggesting that hierarchy or composition of components that interact with each other to fulfil their mutual purpose could be a characteristic of complexity.

Nature needs no understanding of its complexity to marvel at its creation. Complex organisms live, learn and adapt to their environment, passing on their instincts and habits from parent to offspring in their quest for survival. So inept, it is hardly surprising that we have little understanding of how nature really works. Thankfully, through generations of scientific discovery, humans have gained some insight into this remarkable design process, harnessing knowledge of genetics - the building block of life; a basic concept of evolution - life's design process; molecular biology - material for creating life; and neural networks - the essence of intelligence.

The fundamental key to the evolution of every living organism is the heredity transfer of genetic information from parent to offspring through genes. Through the application of this process over millions of years, nature has found a way to create the most complex and precise living organisms.

2.1.1 Adaptation and Natural Selection

Part of the theory of evolution is a concept called 'Adaptation'. This refers to living organisms, and the assets that enable them to survive and reproduce. Darwin [7], demonstrated this concept by illustrating the characteristics of a woodpecker. Its beak allows them to make holes in trees, giving them access to food all year round, and the ability to make nests within their holes. Their long tongue allows them to find insects inside the tree hole for food. Their tail is stiff, and enables them to brace, short legs and long toes can grip the bark. The moulting of feathers leaves the most essential feathers until last.

The catalyst of adaptation is the preservation of useful variation, however slight. Any variation that may help an individual to survive, find food, attract a mate, produce offspring etc., with a higher chance of survival, if captured and passed on to offspring through inheritance will help that species to adapt to its environment. This principle, by which slight variations if useful, are preserved, is called Natural Selection.

In natural selection certain individuals will make more of a contribution to the next generation than others because of their better adaption to the environment. If these attributes can be passed from parent to offspring, then over time they will increase in occurrence and will therefore cause population change to occur.

Individuals of all species must fight for their share of the common resources that they depend on to survive, denying the environment of their destruction. Individuals must compete for resources against members of their own species, as well as from other species. Variations that help to achieve this purpose, are occasionally captured through inheritance, and passed on to offspring. Selection will operate on survival amongst individuals of all genotypes, or those individuals that produce higher numbers of offspring, thereby operating on the basis of fertility. Darwin concluded in his work "On the Origin of Species" that,

> "*each at some period of its life, during some season of the year, during each generation or at intervals, has to struggle for life, and to suffer great destruction. When we reflect on this struggle, we may console ourselves with the full belief, that the war of nature is not incessant, that no fear is felt, that death is generally prompt, and that the vigorous, the healthy, and the happy survive and multiply.*" Darwin, C [8].

2.1.2 Reproduction and Genetics

In the natural world, no two individuals are the same. Identical twins may share the same genetics, but are never exactly alike. This uniqueness comes partly from genes which act as guidelines in determining the way we look and how the body works. Genetics is the branch of biology concerned with studying heredity and variation. The word heredity simply means the transmission of characteristics from parents to their offspring, while variation means the observable differences between all living things.

During reproduction, genes are transferred between parent and child. Chromosomes carry the genes in a linear line up pattern, like a string of beads. The number of chromosomes are different between species as shown in Table 2, however, in mammals it is always an even number. This is because, chromosomes come in pairs, one from

Table 2: A Sample of the Number of Chromosomes Found in Several Species

SPECIES	No. OF CHROMOSOMES
Human	46
Chimpanzee	48
Dog	78
Horse	64
Fruit Fly	8
Pea	14

the mother, and the other from the father. A single parent will only be giving their offspring half of their own genetic recipe.

Genes are made up of acid molecules called Deoxyribonucleic acid or DNA. These molecules consist of chemical units called nucleotides. Nucleotides always come in pairs forming the characteristic double helix; two strands that intertwine with the nucleotide pairs forming links across the middle. From genes, proteins are created via several intermediate stages of which the main two stages are known as transcription, where a copy of the DNA is made using ribonucleic acid (RNA), and translation where the amino acids required to make the protein are fixed together. The amino acid coding for the proteins is what ultimately will enable an adult body to develop from an egg cell, through a process called embryology.

Offspring receive their gene makeup by inheriting genes from each parent via a process called reproduction. Recombination involves a process in which complementary stands of two parental duplex DNA, bind and exchange genes forming hybrid DNA. First, copies are made of each parent chromosome by separating the helix strands of the parental DNA to act as templates for the synthesis of the complement. The parental duplex is replicated from the original DNA to form two daughter duplexes, each consisting of one parental stand from the original unwound DNA helix, and a newly synthesised strand. The two daughter strands are then recombined to form a complementary pair via a process called crossover.

Figure 1, illustrates the crossover process involving two daughter strands where a stretch of Hybrid DNA is formed through the recombination intermediate (when a single strand crosses over from one duplex to the other).

The formation of hybrid DNA, requires the sequences of the two recombining duplexes

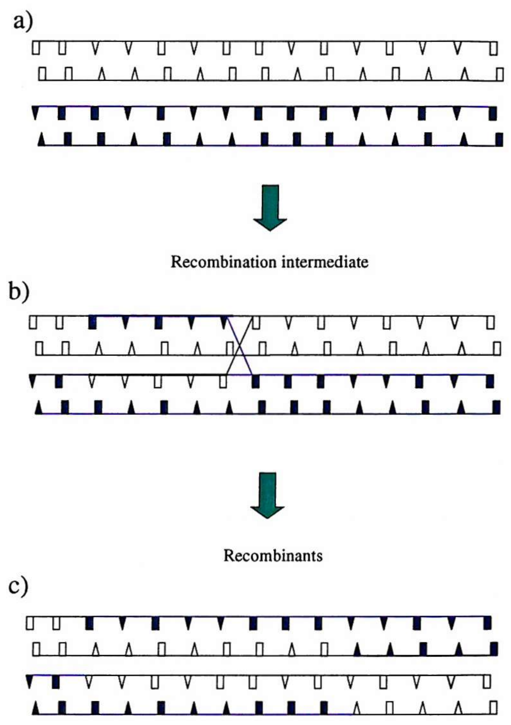

Figure 1: Schematic Illustration of the Crossover process

Table 3: Different Types of Mutation

Type of Mutation	Original Sequence	Mutated Sequence
Subsitution	ATC**G**TTAGGC	ATC**C**TTAGGC
Deletion	ATCG**TTA**GGC	ATCGGGC
Insertion	ATCGTTAGGC	ATCGT**CCA**TAGGC
Inversion	AT**CGTT**AGGC	AT**TTGC**AGGC
Duplication	AT**CGTT**AGGC	AT**CGTTCGTT**AGGC

to be close enough to allow pairing between the complementary strands. Where genetic differences exist, correction is attempted to repair damage to DNA.

During this process of replication, errors may accidently occur, known as mutation. Synonymous or silent mutations are those that occur between two codings for the same amino acids that have no effect on the protein sequence. Non-synonymous or meaningful, point mutations do change the amino acid.

For a more detailed description of genetics and the recombination process, the texts of [9] and [10] are recommended.

2.2 Artificial Evolution and The Genetic Algorithm

Natural evolution is adept at discovering highly precise functional solutions to particular problems posed by an organism's environment. Although the mechanics of evolution are highly complex, several key processes such as natural selection, recombination and mutation, continuously emerge as key components in natures iterative optimisation cycle.

It is quite natural therefore to describe evolution in terms of an algorithm that can be used to solve difficult non-linear engineering optimisation problems. It is this goal that has lead to a relatively new field of computing; Evolutionary Computing.

2.2.1 Evolutionary Algorithms

In general, since evolutionary algorithms are based on a simplification of the organic model, several characteristic properties are generally shared amongst them.

1. Search is a parallel process provided through a collective population of learning individuals. Each individual represents a point in the search space of potential solutions to the problem.

2. Descendants of individuals are generated by stochastic processes based on models of mutation and recombination.

3. Natural selection is modeled to ensure that good genetic traits propagate from one generation to another. This generally involves a stochastic process that mimics survival of the fittest by means of comparing individuals performance in their environment. Such a selection process favours better individuals to reproduce more than those that are relatively worse.

Although most evolutionary computing algorithms follow a similar set of naturally inspired rules or models, two distinct classifications of algorithms have emerged, Genetic Algorithms and Evolution Strategies:

1. **Genetic Algorithms** evolve populations of chromosome structures called Genotypes using genetic operators such as crossover and mutation to adapt the encoding of the structure. Probabilistic selection is used to ensure that highly fit members recombine and produce offspring. When determining the fitness of a Genotype, it is decoded from its genetic form, into its real and useable form called the Phenotype. Typically, GAs can be used for search and optimisation of a wide range of problems by changing the Genotype-Phenotype mapping. Genetic Programming is a subset of GAs that apply evolution directly onto a rule based language that can be used as a computer program.

2. **Evolution Strategies** [1] were originally developed to adapt rule sets by breeding and mutating members of a search space, as well as evolutionary parameters that affect mutation. Mutation is based on normal distributions whose shape

and effect are controlled by the members evolutionary parameters. Typically selection is deterministic. These algorithms differ from Genetic Algorithms as they are based dominantly around mutation and the evolution of controlling parameters, whereas GAs are based around the breeding of members.

Evolutionary Programming embraces normally distributed mutation as the key genetic operator, with probabilistic selection. Evolutionary Programming was originally developed to evolve Finite-State Machines, they seek to iteratively generate increasingly better solutions to a static or dynamically changing environment.

2.2.2 The Genetic Algorithm

The basic principles of Genetic Algorithms are based upon the analogy of natural behaviour. In nature, individuals compete with each other for resources such as food, water and shelter. Members of the same species will also compete with one another to attract a mate. Those individuals that are most successful in surviving and attracting mates, will have a higher chance of producing more offspring and propagating some of their genes into future generations. The propagation of genes from highly adapted or "fit" members forms the basis for evolution, as the combination of good characteristics from parents can sometimes produce "fitter" offspring.

Based upon this analogy, GAs evolve a population of candidates, each representing a possible solution to a given problem. A fitness score is assigned to each individual according to how good that individual is with respect to a given goal. Highly fit individuals are selected to participate in cross breeding with another individual to produce offspring sharing features from each of its parents. Members that are not selected to reproduce die out, and the process is restarted on the new generation of members.

The basic or simple GA as described by Goldberg [11] comprises four important steps illustrated in Figure 2; initialisation, evaluation, reproduction and convergence.

The initial population of chromosomes is created either randomly or by perturbing an input chromosome. How the initialisation is done is not critical as long as the initial population spans a wide range of variable settings (i.e., has a diverse population).

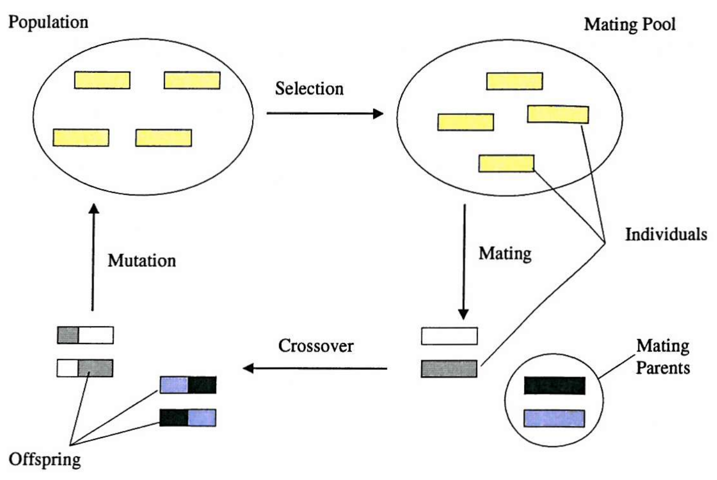

Figure 2: Schematic Diagram of the Main Processes in a Genetic Algorithm

The population of candidates are evaluated and given a fitness score, which should provide a measure of improvement against a pre-defined objective. In single-objective and multi-objective problems, this measure will be based on an evaluation function and then scaled against an average or best fitness within the population.

In the third stage, reproduction, a breeding process is simulated by selecting members of the population to participate in the next generation. These selected members, termed Parents, undergo a breeding process typically using two genetic operators, crossover and mutation. The crossover genetic operator combines aspects of two parent genes to produce two subsequent children genes. This type of gene combination serves to produce children that have a high probability of having a higher fitness than their parents, while maintaining some of the genetic history of their ancestors. In order to maintain diversity in the search space, genetic mutation is introduced into the population. These new members will either be automatically entered into the new generation of candidates to participate further in the evolution process or will be added to the existing solutions pool and undergo further stochastic selection.

The new child gnomes created from the reproduction process are evaluated for their fitness. The parent gnomes die out and no longer participate in the evolution process. The last two steps are repeated until a convergence criteria is met. The basic process

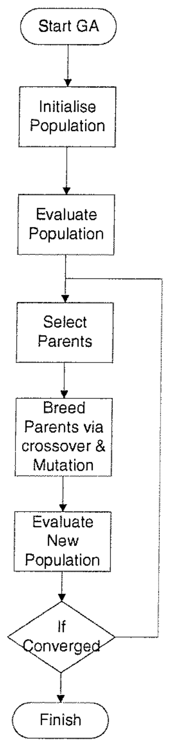

Figure 3: Simplified Flow Diagram for a Simple Genetic Algorithm

for a simple GA is shown in Figure 3.

Why Genetic Algorithms work

Several idea's have been offered as to how Genetic Algorithms are able to effectively explore complex search area's. One such theory that has emerged and received significant attention amongst the evolutionary computing community, is the Schema Theory [12].

Schema Theory considers the chromosome strings as subsets or hyperplanes of the search space. If we take a three dimensional space for example as shown in Figure 4, encoded with the string "000" at the origin, using three bits. The corners are numbered by bit strings with all corners differing by exactly "1" bit. Representing "*" as a wild card match symbol, then the front plane of the cube can be represented

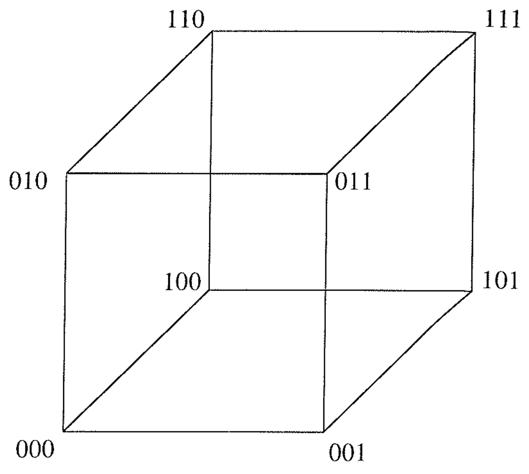

Figure 4: Hypercube Representation of GA Search Space Encoding

by the special string "0**", and each of the other five planes by "*0*", "**0", "1**", "*1*" and "**1". Strings containing "*" are referred to as schemata, and each schema corresponds to a hyperplane in the search space. In a GA, a population of sample points provides information about numerous hyperplanes, with low order (the order refers to the actual number of real bit values represented in the schema) schema sampled by numerous points in the population. In this context, GAs are intrinsically parallel, because they sample many different hyperplanes simultaneously in a parallel fashion, but it is the cumulative effect of sampling a population of points that provides important statistical information about any particular subset of hyperplanes.

By necessitating competitions between the various hyperplanes in parallel, competing hyperplanes increase or decrease their representation in the population according to the relative fitness of the strings that lie in those hyperplane partitions [13]. Recombination through crossover, recycles hyperplane subsets, which can be propagated if good schema are represented and survive disruption from the operation. Mutation acts to generate new schema combinations, adding new hyperplanes to the competition. Forrest and Mitchell [12] illustrate in detail the importance of schema order in GA search, on several theoretical optimisation examples.

Despite this formulation, this theory does not easily extend to all GA observations such as the effect of union crossover [14] involving the exchange genetic information between chromosomes at every other string bit. The schema theory would suggest that this operation would be disruptive to the evolution process. Experiments by Goldberg [15] have indicated that the ordering of the hyperplanes is critical to the success of the search, particularly when GAs are applied to many real problems. These deceptive problems can arise if the schema is ordered incorrectly, such that following samples of the good schema, does not necessary lead the search process towards the optima. The correct ordering of schema is unknown at the start of a given problem. To help encourage correct schema ordering, the Messy GA formulation was introduced [16], which allows the evolution of the ordering of genes along the length of the chromosome structure as well as the genetic information that they contain. Goldberg has shown that Messy GA's are able to solve some problems that exhibit deceptive characteristics better than simple Genetic Algorithm implementations. However their implementation on real problems is difficult and has yet to be achieved.

2.2.3 Searching for the Optima

Mathematical Definition of an Optimisation Problem

An optimisation problem is defined by a set Y and an objective function $f : Y \longrightarrow \mathbb{R}$. Each candidate $y \in Y$ is given a value $f(y)$ by the objective function f. The objective of an optimisation problem $\mathbf{P}_o(Y,f)$ is to find the value $y_{opt} \in Y$ that gives the minimal objective function f, that is $\forall y \in Y, f(y_{opt}) \leq f(y)$. If $P(Y)$ represents the set of subsets of Y, a neighbourhood function $N : Y \longrightarrow P(Y)$ is defined where a candidate y_m that verifies $\forall y \in N(y_m), f(y_m) \leq f(y)$ is called a local optimum. The solution y_{opt} is usually called the global optimum to avoid confusion.

A combinational problem is defined by a finite search space S where the problem is formulated by the set of constraints $C = \{c_1, c_2, ...\}$. A candidate that satisfies all constraints c_i of a combinational problem $\mathbf{P}_c(S, C)$ is said to be a feasible solution of $\mathbf{P}_c(S, C)$ (an infeasible solution otherwise). The objective of a combinational optimisation problem $\mathbf{P}_{co}(S, C, f)$ is to find s_{opt} where $s_{opt} \in X$ and $f(s_{opt}) = min_{s \in X} f(s)$ where $X \subseteq S$ of all feasible solutions $\mathbf{P}_c(S, C)$.

Global and Local Search Techniques

There are many methods available to engineers to search a given landscape for an optimum. The success and practicality of these methods, are usually dependent on the complexity of the problem, the modality of the landscape to be traversed and the availability of gradients. A full overview of the different approaches available is beyond the scope of this work, a comprehensive overview of the optimisation problem and optimisation techniques is offered by Bishop [17] and Siddal [18]. This section is a generalisation of approaches used.

In searching a solution space, a tradeoff must be established between two conflicting objectives: exploiting the information contained in the previously obtained solutions, and exploring new regions. Pure exploitation makes exclusive use of existing information; pure exploration abandons all known solutions on the premise that better solutions exist elsewhere. Hill climbing, or perturbation routines are good examples of exploitative methods, where the best known solution is always used as a starting point for improvement. Such techniques take a very narrow view of where opportunities for improvement lie; they are extremely dependent upon their starting locations and are susceptible to becoming trapped at local optima. At the other extreme, totally random search processes provide excellent exploration but make no use whatsoever of acquired knowledge. Random search is a good example of such a process.

An example of using exploitative search methods is given in Figure 5. Suppose for this one dimensional problem $f(x)$, we wish to use the derivative with respect to the design parameter x to indicate the best direction of search. Using Taylor series expansion, an approximation for the derivative is given as

$$f^{'}(x) \approx \frac{f(x+h)-f(x)}{h} + O(h^2 f^{''} + h^3 f^{'''} \ldots), \tag{1}$$

where h is the step size. However to implement any approach that uses such a gradient, N evaluations of the objective function will be required for each sample point, where N is the number of design variables. This can not only become expensive if N is large, but Equation 1 may also include roundoff and truncation error due to the finite size of h, and truncation of higher order terms. If the problem domain is

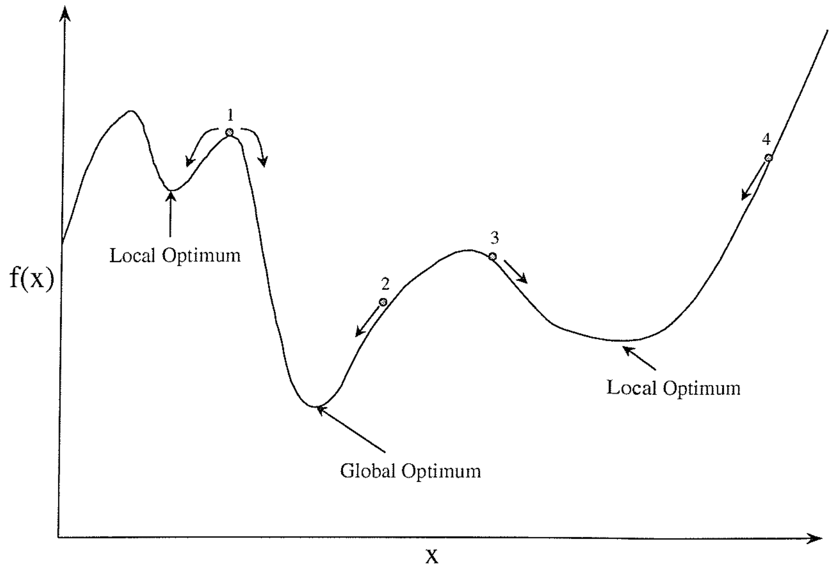

Figure 5: Search Space Representation Featuring Several Local Optima's

constrained, then a derivative with respect to each constraint will also be required. In the relatively simple problem given in Figure 5, the starting point of the search will also play an important factor to the success of locating the global optimum. In this illustration, starting at point 2, will provide a gradient that would indicate the appropriate direction of the global optimum. However, starting at points 3, or 4 would indicate the direction to a local optimum, requiring many restarts to achieve the goal using this approach. Implementations of other similarly based methodologys such as 'divide and conquer' algorithms and Powell's Direction Set Method [19], will also suffer similar consequences.

In an effort to strike a balance between these two extremes, various techniques have been developed that process information on many candidate solutions within each search iteration. The Simplex method [20] is such an attempt to achieve this balance by setting up n+1 points enclosing a finite n-dimensional volume called a simplex. The Simplex method gropes towards the optimum by flipping, expanding or contracting the simplex. The logic used to operate the simplex is determined by the evaluation at each corner, and always moves in the direction of improvement. Although this method samples $N + 1$ points in parallel, it is still prone to local optima traps. A more explorative technique is the method of Simulated Annealing [21], which applies random perturbations to a search point. Simulated Annealing is based on a principle of slow decent inspired by the annealing nature of metals. Newly formed solutions are retained if they improve the solution, and sometimes if they do not depending on a Boltzman probability function. An annealing temperature parameter is used to control the effect of the Boltzman function. By lowering the annealing temperature, the likelihood of retaining weaker solutions reduces. Typically the annealing temperature is reduced on several occasions during the search process.

Lately, evolutionary computing techniques have been employed to cast a better balance between exploitation and exploration by learning from processes found in nature. Such techniques which generally fall under the Artificial Intelligence umbrella, mimic natural processes such as evolution and genetic theory to simultaneously exploit important information obtained from multiple solutions. Stochastic processes are added to the mechanics of the methods introducing a variable degree of exploration.

With any optimisation approach in complex multi-modal landscapes, a consideration of objective function evaluation cost will often determine the type of approach taken. A trade off must often be made between restarting a cheaper gradient based method many times, and using a more expensive but more robust stochastic approach such as Simulated Annealing or Genetic Algorithm.

2.3 Approaches to Aerodynamic Shape Optimisation

Typically, two approaches have been taken in automated design involving CFD. Indirect methods [5] offer significant savings in computational cost, through iteratively redesigning airfoil shapes in the direction of achieving a pre-described pressure or velocity distribution. Such methods generally involve the use of calculated derivatives of the objective function, making the process fast enough for day to day design and analysis. The main drawback to this approach, is that the objective may be infeasible (an airfoil design prescribing the required pressure distribution may not exist). Also, if drag is a component of the objective, then attempts to define a pressure distribution that minimises C_D, can lead to poor solutions. The evaluation of pressure drag from

$$C_{D_p} = \oint C_p d\bar{y} \tag{2}$$

is typically swamped with numerical noise.

Direct shape optimisation attempts to optimise directly against the objective function, such as requiring minimum drag. This more expensive approach either requires the calculation of cost function sensitivities for use with gradient based optimisers, or employs black-box type optimisation tools that can work directly with the design variables and the cost function. Examples of these would be the Simplex Method, Powell's Direction Set method [19], Simulated Annealing, and Genetic Algorithms. The more advanced tools such as GAs and Simulated Annealing, are generally more robust to local optima and noise, and are well suited to preliminary design investigations rather than the fine tuning of detailed designs. Because they require large numbers of design solutions, such methods are typically used with less expensive CFD tools such as Panel Method implementations.

Preliminary investigations into the use of sensitivity analysis with CFD solvers by Hicks and Henne [22], found that the solution of cost sensitivities by perturbing the design space with respect to each design variable, was too expensive for use with high fidelity CFD solvers. A more suitable solution based on Control Theory described by Jameson [23] [24][25], achieved the solution of cost sensitivities via an adjoint formulation of the Cost function and Flowfield Solution.

If the governing flow equation R expresses the dependence of the flow variables w on the physical boundary F, this can be written as

$$R(w, F) = 0, \tag{3}$$

then a change in F results in a change

$$\delta R = \left(\frac{\partial R}{\partial w}\right) \delta w + \left(\frac{\partial R}{\partial F}\right) \delta F = 0. \tag{4}$$

If the cost function I is determined by the change in flow w, due to a modification of boundary F, such that

$$I = I(w, F), \tag{5}$$

then a change in F results in a change

$$\delta I = \frac{\partial I}{\partial w}\delta w + \frac{\partial I}{\partial F}\delta F. \tag{6}$$

Using control theory, the flow equation is considered as a constraint in such a way that the cost equation does not require multiple flow solutions. This is achieved by eliminating δw. By introducing a Lagrange Multiplier Ψ, we have

$$\delta I = \frac{\partial I}{\partial w}\delta w + \frac{\partial I}{\partial F}\delta F - \Psi \left[\left(\frac{\partial R}{\partial w}\right)\delta w + \left(\frac{\partial R}{\partial F}\right)\delta F\right], \tag{7}$$

which can be rewritten as

$$\delta I = \underbrace{\left[\frac{\partial I}{\partial w} - \Psi\left(\frac{\partial R}{\partial w}\right)\right]\delta w}_{Field} + \underbrace{\left[\frac{\partial I}{\partial F} - \Psi\left(\frac{\partial R}{\partial F}\right)\right]\delta F}_{Boundary}. \tag{8}$$

By choosing Ψ to satisfy the adjoint equation

$$\left(\frac{\partial R}{\partial w}\right)\Psi = \frac{\partial I}{\partial w}, \tag{9}$$

the gradient is defined through

$$\frac{\delta I}{\delta F} = \frac{\partial I}{\partial F} - \Psi\left(\frac{\partial R}{\partial F}\right). \tag{10}$$

Through this approach, the gradient can be determined indirectly by solving an adjoint equation which has coefficients determined by the solution of the flow equation. The basic implementation process is given in Figure 6.

The cost of solving the adjoint is comparable to that of solving the flow equation. Hence a single design iteration will cost approximately the cost of two field solutions. In Jameson's implementation, approximately 100 design iterations were required to

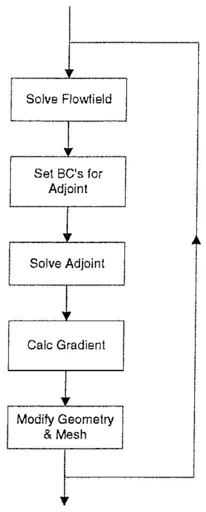

Figure 6: Process for the calculation of cost sensitivities via the adjoint formulation

achieve convergence. Burgreen and Baysal [26] implemented an adjoint formulation of the three dimensional Euler fluid equations using a Bezier parameterisation for the wing. A novel approach to three dimensional adjoint formulation on unstructured meshes was implemented by Cross [27], by coupling a structured mesh based adjoint formulation to an unstructured flow solution through interpolation. Such an approach can facilitate the use of optimisation on complex geometric configurations with advanced flow solvers.

2.3.1 The Application of GAs to CFD Optimisation

The global search capabilities of Genetic Algorithms have recently been introduced to aerodynamic design problems. Gage [28] examined the possibility of a role being played by a Genetic Algorithm in preliminary aircraft design. Gage found the tool highly robust and effective to apply in the design of a wing planform for minimum induced drag, but commented on the large computational time required while using simplified analysis theory. Studies into the role of Genetic Algorithms in the conceptual design of aircraft has also been made by Crispin [29]. A step towards aerodynamic shape optimisation was reported by Yamamoto [30] who applied a Genetic

Algorithm to a 2D wing section design for maximum Lift to Drag ratio (L/D) using a Navier-Stokes code parallelised on a 32 Node Super-Computer. Although the optimisation schemes proved successful in 2D wing optimisation, little was known about the modality of the objective surface or how a Genetic Algorithm would compare with other much credited methods. Obayashi and Tsukahara [31] attempted to address this issue by first examining the objective surface and then comparing three very different optimisation methods based on, Gradient Method, Simulated Annealing, and a Genetic Algorithm. The airfoil section was represented as a linear function of four cubic splines. Obayashi and Tsukahara examined a two variable aerofoil design problem, and found the objective surface to be peaky and convex. This problem was deemed to be unsuitable for Gradient Methods, whose solution is reliant on the initial starting point. The surface was improved however through the introduction of a penalty function to constrain the foil thickness, which in effect reduced and concentrated the number of peaks to a much more attainable problem domain. The comparison of the three optimisers showed a marked increase in design performance in the case of the Genetic Algorithm. Analysis into the starting point was made, showing that the performance of the Gradient Method and Simulated Annealing implementation were highly dependent on the initial starting point. In contrast, the performance of the Genetic Algorithm was considered to be independent of the starting population. The CPU cost of using the Genetic Algorithm was significant, almost 80 times that of a single use of a Gradient Method. This was however deemed to be an acceptable cost for this type of optimiser since it resulted in a near twofold gain in performance, while maintaining a good level of robustness. Application of Simulated Annealing methods has been further approached by Aly *et al.* [32] achieving significantly improved performance over a Gradient Method at comprisable cost. His application however was significantly hampered by local optima.

There has been significant activity within the literature over the past three years involving the coupling of Genetic Algorithms to CFD solvers. Poloni [3], Périaux *et al.* [33], and Holden [2] have successfully shown that good aerofoil shapes can be found with Genetic Algorithms. Doorly [34] has highlighted concern over the modality of the objective surface for airfoil design problems, with GAs tending to converge quickly, and finding different solutions on each search. The quality of the geometric representations used with GAs has also been questioned by Reuther and Jameson [35], and Holden [36], who have found that some representations are more suited to GA design than others, but may not necessarily provide the high level of geometric sensitivity required for aerodynamic design. Lépine *et al.* [37] demonstrated that geometric accuracy can be found using Non-Uniform Rational B-Spline (NURBS) representations, but its parameterisation would require over 13 control points, requiring a total of 52 design variables for a single 2D Section.

2.4 Representation and Parameterisation of Shape for Evolutionary Design

The expression of the shapes we seek to design, plays a most important role in design optimisation. In evolutionary approaches, Genetic Algorithms can be used to adapt a parameterised encoding called a genotype, using crossover and mutation operations. This encoding can then be mapped to the final shape called a phenotype, through the use of geometric representation. The mapping between genotype and a geometric phenotype can be divided into two separate processes, representation of shape from a set of parameters, and the parameterisation of these parameters into the design variables used to form the resultant chromosome.

2.4.1 Airfoil Representation and Parameterisation in Shape Optimisation

For a numerical optimisation process, a surface representation is sought that will offer a detailed description of a surface, with an enriched design space whilst maintaining the smallest number of parameters to be defined. The descriptions of two-dimensional curves is well known, and several techniques are readily used throughout the engineering optimisation community [2]. Such techniques have been extended to simple three-dimensional objects using chordwise sections on a known or set planform. Whilst this method has been effective for a small number of applications, it only offers a limited scope for novel design search. The ability to define accurate three-dimensional surfaces is an essential component of an automated design process. The construction and definition of geometry in engineering is a vast and complex problem. Most engineers use dedicated Computer Aided Design (CAD) facilities for the accurate development of 2D and 3D drawings. However these sophisticated, complex programs usually require thousands of input parameters for the complete definition of most modern designs, and are inadequate for design optimisation [35]. In aircraft design for example, the accurate definition of a single airfoil section may require several hundred points to be defined in three dimensional space.

For the representation of airfoil sections, approaches taken generally fall in to two categories, Shape Functions, and Curve Interpolations. Parametric representations are taken as an exception to these categories. Parametric methods are considered beyond the scope of this work, and are more suited to preliminary investigations into target design specification, rather than to accurate shape optimisation. An example of parametric application to airfoil design was given by Giunta and Sobieski [38] who used a parametric definition for airfoil camber and twist in their aeroelastic analysis of a supersonic transport aircraft.

Shape Functions

Basis Vectors [39][40], express new design points through linear combinations of base shapes or functions through the generalised form

$$\bar{\mathbf{R}} = \bar{r} + \sum_i \bar{u}_i \bar{U}_i \tag{11}$$

where $\bar{\mathbf{R}}$ is the design shape, $\bar{r}$ is the baseline shape, $\bar{u}_i$ are the basis functions/vectors, and $\bar{U}_i$ is a vector of scalar weights or design variables. In some applications of this approach to airfoil design, airfoil shapes themselves have been used as basis vectors [31], leading to a significantly reduced parameterisation ($\bar{U}_i$). However, this approach would be restrictive in novel design and flexibility since all new design points offered by this approach are indirectly defined by the given basis vectors. The choice of airfoil basis vectors is critical to the success of implementation, with the additional danger that each basis shape could represent locally determined optimum if chosen as a result from previous designs and optimisation analysis.

A similar but more flexible approach which has become popular in CFD applications, is the use of analytical functions to provide basis vectors ($\bar{u}_i(x)$), which are applied to a base shape $\bar{r}$. Hicks and Henne [41] demonstrated the suitability of this method with their compact set of 'Bump' functions

$$b(x) = \left[\sin\left(\pi x^{\frac{\log(0.5)}{\log(t_1)}}\right)\right]^{t_2}, 0 \leq x \leq 1 \tag{12}$$

where t_1 and t_2 control the intensity and location of the bump. This method was implemented by Reuther and Jameson [35] using 25 functions. Hager et al. [42], Lee and Eyi [43], and Elliott and Peraire [44] used 10 shape coefficients based on different shape functions. The flexibility of this method may be limited through both the base shape $\bar{r}$ and modification scalars $\bar{U}_i$. Further refinements to this method have been made by Hicks and Vanderplaats [22], introducing a second set of basis functions to offer increased sensitivity to assist in the presence of shockwaves. This method can also be extended to three dimensional shapes to a limited degree as demonstrated by Elliott and Peraire [45].

The value of shape functions can be enhanced by finding an orthogonal set. The use of orthogonal shape functions in the representation of airfoil sections was adopted by Kuruvila et al. [46] based on the NACA four series. Chueng [47] and Drela [48] also demonstrated the effectiveness of this method based on a sinusoidal series.

2.4.2 Interpolation Functions

In early studies involving gradient optimisers with CFD, the computational domain grid points themselves were used directly in the parameterisation [49]. This approach is easy to implement and geometry changes are limited only to the number of discrete points. However it is difficult to maintain a smooth geometry and a significant presence of noise is usually present in the final design.

The use of polynomial and spline representations are popular in CAD [50] and are well suited to automated design. A polynomial can describe a curve in a very compact form with a small set of design variables, with many of the curve noise problems encountered with discrete approaches automatically smoothed out as pointed out by Braibant [51], Reuther and Jameson [35]. If the curve is parametric, almost any curve or surface could potentially be described with the use of polynomial and spline interpolation techniques.

The simplest description using such an interpolation approach, is a polynomial such as

$$\bar{\mathbf{R}}_g(u) = \sum_{i=0}^{n-1} \bar{c}_i u^i \tag{13}$$

where n is the number of design variables, and u is the parametric length along the curve. The $\bar{c}_i$ is a set of coefficient vectors corresponding to the three dimensional coordinates u^i.

The Bezier representation is another mathematical form for representing curves and surfaces. The general form of the Bezier curve may be written as

$$\bar{\mathbf{R}}_g(u) = \sum_{i=1}^{n} \bar{\mathbf{P}}_i \mathbf{B}_{i,p}(u) \tag{14}$$

where n is the number of control points (design variables), and the $\mathbf{B}_{i,p}(u)$ are degree p Bernstein functions. The $\bar{\mathbf{P}}_i$ are the control points (creating the control polygon), and are typically used as design variables. The Bezier form is far better than the power basis of Equation (13) with the control points relating to the curve much more closely. The convex hull enclosed by joining the control points together, contains the resulting curve which is a very useful property when defining the geometric constraints.

The Bezier form is highly efficient in representing simple curves, however more complex curves require a high-degree Bezier form which increases the round-off error. A composite form of the Bezier curve which uses several low-degree Bezier segments is the B-spline described by

$$\bar{\mathbf{R}}_g(u) = \sum_{i=1}^{n} \bar{\mathbf{P}}_i N_{i,p}(u) \tag{15}$$

where $\bar{\mathbf{P}}_i$ are the B-spline control points, p is the degree, and $\mathbf{N}_{i,p}(u)$ is the i-th B-spline basis function of degree p. Reuther and Jameson [35] used 18 control points to define a fourth order B-Spline to represent an airfoil section, but found that the resultant solution contained to many waves on the surface to be useful for aerodynamic design. Similar findings have also been reported by Doorly *et al.* [52], and Yamamoto and Inoue [30]. The only real geometric drawback to the regular B-spline is their inability to represent implicit conic sections accurately. However a special form of the B-spline, the Non-Uniform Rational B-Spline (NURBS), can represent most parametric and implicit curves and surfaces without loss of accuracy as illustrated by Ventura for the representation of complex hull geometries [53]. A NURBS curve is defined as

$$\bar{\mathbf{R}}(u) = \frac{\sum_{i=1}^{n} N_{i,p}(u)\, W_i \bar{\mathbf{P}}_i}{\sum_{i=1}^{n} N_{i,p}(u)\, W_i} \tag{16}$$

where the $\bar{\mathbf{P}}_i$ are the control points, W_i are the weights, and the $N_{i,p}$ are degree p basis functions. With this representation, both the control points $\bar{\mathbf{P}}_i$ and their corresponding weights W_i can be used as the design variables.

2.4.3 Three Dimensional Description of Aerodynamic Bodies

B-spline based methods are probably the most popular representations used in automated design today [54][55]. Their popularity is mainly due to their ease of implementation and smooth results, but also because such techniques can be integrated with suitable design databases found on many CAD systems. However, despite recent progress in using these representations, it is still difficult to parameterise and construct accurate, complex three-dimensional models for use in optimisation based solely on polynomial and spline representations.

Generally, $2\frac{1}{2}$D approaches are often employed in 3D surface descriptions, defining the surface as an interpolation across several different 2D sections. Through this approach, many of the techniques highlighted in Sections 2.4.1 and 2.4.2, can be used for the sections. One novel approach to the complete description of a surface is the treatment of surface generation as a boundary-value partial differential equation problem [56]. Using this technique it is possible to represent a geometry such as an aircraft wing or ship hull form, with a compact set of design variables. Included with this definition are surface grids, volume grids, and sensitivity derivatives. Parameterisation using PDE's for complex bodies is however time consuming and can only parameterise surfaces [57][58].

Separate attempts have been made at designing the geometric planform alone, based on the $2\frac{1}{2}$D approach. Gage [59] applied Genetic Algorithm's to the problem of wing planform optimisation for minimal induced drag, using the computational panels themselves as the representation. In Gage's unique parameterisation of the representation, the complexity of the panels were adapted [28], by allowing arbitrary panel encodings to crossover with one another. For instance, the panel representing a winglet, could be crossed with a centre wing panel. Although this parameterisation would be difficult to apply to more precise geometric forms, Gage was able to evolve quite unique planforms demonstrating the ability for a Genetic Algorithm to navigate the most complex search spaces. Other, more traditional approaches to planform optimisation can be found in [60].

There are significant similarities in the parameterisation problems encountered in 2D and 3D representations. To ensure accurate and highly evolved geometric definitions can be obtained, a large number of parameters are required. Some of the geometric representation schemes, such as the use of basis vectors, or Bezier interpolation functions, offer a relatively small parameterisation and hence search space to the optimiser. However, to extend optimisation towards more detailed geometric definition, the use of more sensitive techniques such as NURBS representations must be adopted. So far, similar attempts with BSpline interpolation with Genetic Algorithms has proved difficult due to the large number of design variables required for their definition, surface waves that are often encountered through their adoption.

2.5 Evolving Geometric Complexity

Almost all applications of GAs to shape optimisation problems are based on fixed parameterisation of the landscape domain. Genetic Programming [61] has been successful in the application of evolving Lisp programs, by allowing adaption of the parameterisation structure. Yamamoto *et al.* [30], applied a similar concept to adapting the number of parameters used to define the curvature of an airfoil, by representing the B-Spline nodes via radius and angle expressions relative to another node. By including a knowledge of geometry into the chromosome encoding, they were able to add and remove new nodes from the chromosome structure in a similar way as adding or deleting structures and expressions from a Genetic Program. Gage [62] adopted a similar process to evolving the topology of wing shapes by allowing the GA to adapt both the basic shape as well as the complexity of defining parameterisation. Both of these implementations allowed the crossover process the freedom to randomly combine very different structures together, in a fashion that could be compared to a reproduction process trying to combine the structure of a leg with one from a hand. Bentley [63] highlighted concern for the large number of lethal mutations that could arise through such crossover processes, and proposed a more restrictive hierarchical crossover operator. One important feature of Bentley's implementation, was that new genes added to the representation were seeded with values such that the resultant phenotype remained as unchanged as possible. For primitive types, new gene values were interpolated from the existing shape, as illustrated in Figure 7. Hierarchical crossover was successfully applied to evolving Lego like geometric structures, and could be extended to other geometric descriptions such as spline methods.

The process for automated design is generally in the form shown in Figure 8, treating each sub process such as design analysis, and geometry definition, as separate entities. How information is passed between entities is not important in this analysis, but the use of formatted files is normally adopted in both the manual and automated implementations. For this work, the automated design process will be represented by the Genetic Algorithm, the geometric representation libraries will replace the geometry generator, and CFD tools will be used for airfoil performance analysis. File stores will generally be used for maintaining a database of solutions and designs.

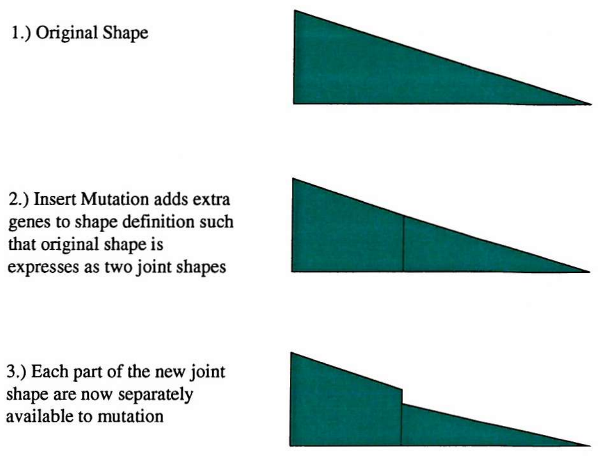
1.) Original Shape
2.) Insert Mutation adds extra genes to shape definition such that original shape is expresses as two joint shapes
3.) Each part of the new joint shape are now separately available to mutation

Figure 7: Mutation of a Primitive Type

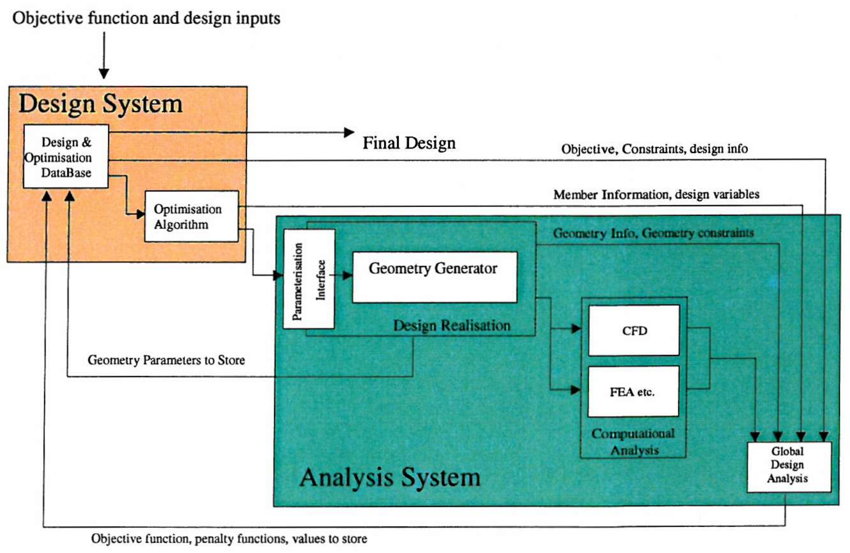
Objective function and design inputs
Design System
Design & Optimisation DataBase
Optimisation Algorithm
Final Design
Objective, Constraints, design info
Member Information, design variables
Geometry Info, Geometry constraints
Parameterisation Interface
Geometry Generator
Design Realisation
Geometry Parameters to Store
CFD
FEA etc.
Computational Analysis
Global Design Analysis
Analysis System
Objective function, penalty functions, values to store

Figure 8: Automated Design Process

2.6 Summary

Evolutionary algorithms are expensive and slow, requiring large numbers of new designs to be sampled. Genetic Algorithms are most suited to explorative aerodynamic design rather than improving existing ones, because new candidates are sampled in parallel at each iteration which requires a large and diverse pool of candidate designs. GA evolution of aerofoil shapes to arbitrary detail is hampered by several key problems. The implementation of the Genetic Algorithm with the flow solver can restrict its ability to adapt aerodynamic shapes due to range of geometries offered by the fixed chromosome encoding used. The evolution process can become trapped in local optima, converge too quickly, or suffer from high levels of bad DNA encodings within the population, thus weakening candidate diversity. Increased robustness, and tolerance to local optima could improve the Genetic Algorithms ability to adapt shape. Some form of genetic repair of bad DNA would also improve evolution efficiency.

A wide selection of geometric representations have been used in the literature to define airfoil and wing shapes. From this wide selection, only Bezier curves and some basis shape approaches appear to be able to offer smooth optimised shapes when used with a relatively small parameterisation. Some instances in the successful use of the more sensitive B-Spline and NURBS methods have been recorded, but these have generally required a much larger parameterisation, and would not be suitable if extended to three dimensional wing-body design. A more thorough investigation into the interaction of different geometric representations with Genetic Algorithms may offer some indication of how this problem should be approached, especially if further adaptation and refinement is sought.

A hierarchical genetic encoding of both representation and parameterisation may allow for increase in shape adaptivity by including the complexity of the parameterisation into the genetic encoding for adaption. Hierarchical crossover and representation aware mutation operators, have been used successfully to adapt the complexity of primitive objects in simple designs. The application of this approach to more sensitive representations such as B-Spline curves should be investigated.

This work describes the development of a framework for evolving geometric structures to arbitrary detail. The scope of this approach will encompass the problem of defining simple curve shapes capable of describing aerofoil sections to a high degree of detail, and their adaptation to a given problem via a process of simulated evolution.

Chapter 3

Evaluation of Fluid dynamic Characteristics

Automated design encompasses the practice of iteratively searching for a design that meets some predefined criteria without human intervention. The manual design process is well known and practiced to a finite degree. The automation of this process is highly sought after, offering many potential savings in cost and time, as well as substantial gains in the final performance of the design.

The application of the automated design philosophy has been made available to engineering design by the recent swing seen in the past few decades from using experimental based analysis of design performance, to computational analysis using theoretical, analytical and first principle based methods. The recent advances in computer hardware and software resources have allowed the use of Computational Fluid Dynamics (CFD) [64] in automated design. Previously, the use of CFD in iterative design cycles required expensive and often specialised computing facilities [65].

The ability to use these solvers in an automated design environment can offer considerable cost, as well as performance benefits, to the design management process. If applied early in the preliminary or conceptual design phase, resources can be diverted in the later design stages to other important design areas, while under the assurance that the design is close to its final description. The adoption of this idea could also be of great benefit in the scoping of contractual proposals as the conceptual design is already close to its maximum performance.

3.1 Ducted Thruster Units

One problem investigated in this work, is a section shape sought for use in a thruster unit suitable for deep sea Tethered Unmanned Underwater Vehicles (TUUV). These units are commonly used for operations such as oil rig maintenance, and operate mainly at depths where cavitation will not occur.

Ducted thruster units offer several efficiency advantages over traditional propeller arrangements by offering a lower propeller loading, by using the duct to draw a larger volume of water into the propeller. Additional thrust is also generated due to the acceleration of flow over the duct. An advantage offered by the arrangement of the duct, is the duct wall proximity to the blades reduces the effect of tip vortices.

In an attempt to eliminate this vortex drag, a ring propeller arrangement has been proposed where the propeller is mounted on a thin ring that sits flush within the external duct. This arrangement tends to lack the efficiency offered by traditional ducted propeller units using a much simpler arrangement with a small gap between the propeller and duct to reduce tip vortices. The main drawbacks to the ring propeller arrangement are mainly the mechanical problems of power transmission, seals and centrifugal bearing problems associated with the new designs.

Remote Operated Vehicles (ROVs), or Tethered Unmanned Underwater Vehicles (TUUVs), require propulsion units that offer high efficiency while relatively light weight and good flexibility for the positioning of the thrusters. Such units currently use small electrical units driving the shaft of the propeller, and are mounted from the nozzle using a 'spider' type bracket. However one drawback to this arrangement is that the position of the motor disrupts the flow of water into the propeller.

A natural progression, drawn from the basis of ducted ring-propellers, has been the idea of tip-driven propellers (TDPs). This concept involves having either a mechanical drive or electrical motor encased within the duct walls. The use of an electromagnetic drive with the absence of physical contact between the drive system and propulsor, offers significant advantages to the mechanical layout which still involves issues such

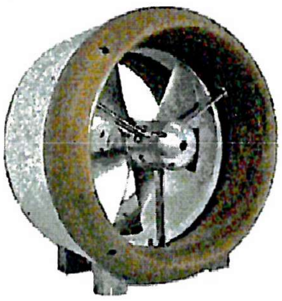

Figure 9: Ducted Thruster Unit

as sealing problems associated with the mechanical drive.

The hydrodynamic performance of a TDP shown in Figure 9 was investigated by Hughes [66] for a unit using a highly efficient permanent magnetic motor design. The efficiency performance still lagged that of traditional ducted units and it was proposed that the unit should be hydrodynamically optimised to examine whether such performance could be recovered. One of the non-optimised and novel features of the tested unit, was the bi-directional characteristics of the section used for the propeller. This bi-directional characteristic was added to enhance the positioning efficiency and effectiveness required by TUUVs.

3.2 Characteristics of Propellers

When a propeller is rotated, a torque is applied on the propeller by the fluid which acts in opposition to the rotation force resulting in a loss in angular momentum. In addition, a thrust is produced by the displacement of fluid by the propeller, causing an increase in rearward momentum.

The non-dimensional coefficient of Thrust K_T is defined as

$$K_T = \frac{T}{\rho n^2 D^4} \tag{17}$$

and the coefficient of Torque K_q is defined as

$$K_q = \frac{Q}{\rho n^2 D^5} \tag{18}$$

where n is the number of revolutions per second turned by the propeller and D is the propeller diameter.

The efficiency of the propeller is defined as the ratio of thrust power produced by the propeller to the input power and is defined as

$$\begin{aligned} \eta &= \frac{Power_{out}}{Power_{in}} \\ &= \frac{K_T \rho n^2 D^4 \cdot V}{K_Q \rho n^2 D^5 \cdot 2\pi n} \\ &= \frac{1}{2\pi} \frac{K_T}{K_q} J \end{aligned} \tag{19}$$

where J is the propeller advance ratio

$$J = \frac{V}{nD} \tag{20}$$

and $\frac{V}{n}$ is the distance advanced by the propeller per second.

In an effort to improve the thrust and torque performance of a propeller, the analysis and design of a blade section taken at 70% propeller radius will be considered via section shape optimisation.

3.3 Description of Aerodynamic Forces

The components of Lift and Moment in relation to the free stream are shown in Figure 10. These basic forces often form the basis for objective comparisons between designs in section shape optimisation. A summary of the basic fluiddynamic coefficients used for comparison are given below.

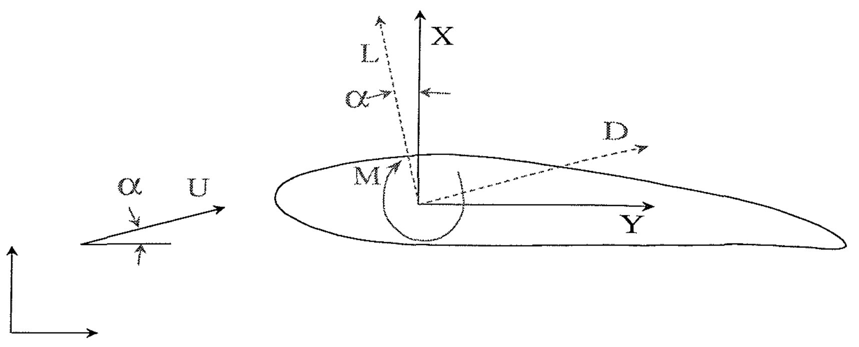

Figure 10: Components of Forces acting on an Airfoil

3.3.1 Lift

Lift is generated by a foil (aerodynamic or hydrodynamic) through the generation of a pressure distribution around the upper and lower sections. The nature of the distribution is determined by a non-linear relationship between the shape of the foil, and the physical characteristics of the applied flow-field (direction, temperature, density, entropy, etc.). For the illustrative pressure distribution given in Figure 11, lift is generated by the suction of the combined upper surface pressure, overwhelming that of the lower surface.

The net lift produced by such an airfoil at zero angle of attack is given by

$$L = \int^{c} \left[-(p_U - p_0) + (p_L - p_0) \right] dx \tag{21}$$

where p_U, p_L are the respective upper and lower surface pressures over an element dx. The non-dimensional form of lift, *lift coefficient*, is given by

$$C_L = \frac{L}{\frac{1}{2}\rho V^2 c} \tag{22}$$

where c is the chord (unit length) of the airfoil. This can also be written as

$$C_L = -\int_0^1 (C_{p_U} - C_{p_L})\, d\left(\frac{x}{c}\right) \tag{23}$$

since by definition,

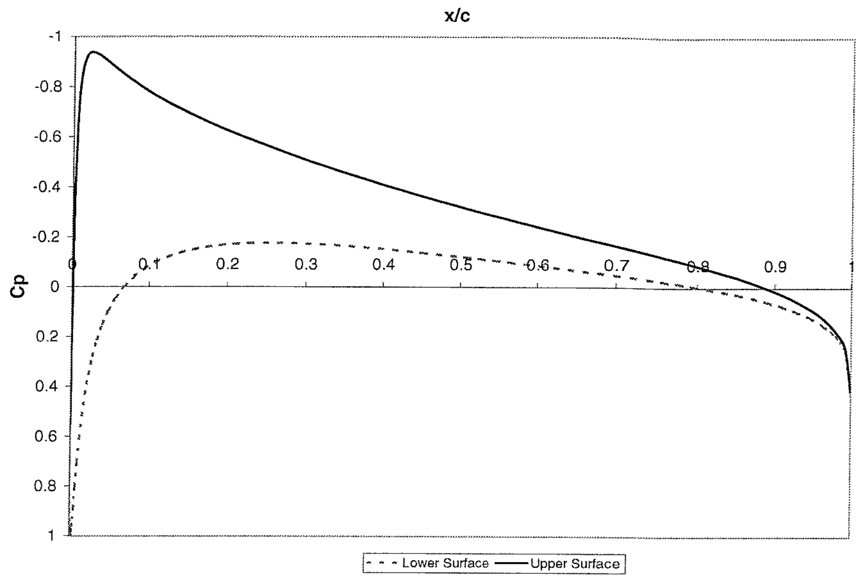

Figure 11: Upper and Lower Surface Pressure Distributions for a NACA0012 Airfoil at $C_L = 0.6$

$$C_p = \frac{p - p_0}{\frac{1}{2}\rho V^2}. \tag{24}$$

For an airfoil at an angle of attack α, lift is defined in a direction perpendicular to the air direction.

Here,

$$C_Z = -\int_0^1 (C_{p_U} - C_{p_L})\, d\left(\frac{x}{c}\right) \tag{25}$$

and

$$C_X = \int_{z_1/2}^{z_2/c} \Delta C_p d\left(\frac{z}{c}\right) \tag{26}$$

where C_X, C_Z are the coefficients of forces acting in the x and z directions.

The coefficient of lift is given by

$$C_L = C_Z \cos\alpha - C_X \sin\alpha. \tag{27}$$

3.3.2 Drag

Pressure drag is the resistance to the flow due to the pressure differential in the flow direction. The pressure drag coefficient may be calculated from the force coefficients by

$$C_{d_p} = C_Z \sin\alpha - C_X \cos\alpha. \tag{28}$$

The total drag force acting on a lifting body (Net force acting against the free stream flow), is described by the force corresponding to the rate of decrease in momentum, in a direction parallel to the undisturbed stream of the external flow around the body. This decrease in momentum, is usually calculated between sections at infinite distances upstream and downstream of the body.

For low speed two dimensional flow over aerofoil sections, the dominant drag force component is known as profile drag and is defined as:

$$C_{D_{profile}} = C_{d_p} + C_{d_v} \tag{29}$$

where C_{d_v} represents the effect of viscous forces over the aerofoil.

3.3.3 Pitching Moment

The pitching moment may also be calculated from the pressure distribution. Around the Ox, Oz axis, the pitching moment due to $Z - force$ is

$$M_Z = \int^{c} \left[(p_U - p_0) - (p_L - p_0)\right] x dx \tag{30}$$

since by definition

$$C_M = \frac{M}{\frac{1}{2}\rho V^2 c^2} \tag{31}$$

the pitching moment about the $Z - force$ is given by

$$C_{M_Z} = -\int_0^c \delta C_p \left(\frac{x}{c}\right) d\left(\frac{x}{c}\right). \tag{32}$$

Similarly, the contribution to C_M due to the $X - force$ may be obtained as

$$C_{M_X} = \int_{z_1/c}^{z_2/c} \delta C_p \left(\frac{z}{c}\right) d\left(\frac{z}{c}\right). \tag{33}$$

Finally the total pitching moment coefficient is

$$C_M = C_{M_X} + C_{M_Z}. \tag{34}$$

3.4 Computational Analysis Techniques and the Panel Method

Computational Fluid Dynamics (CFD) considers the numerical evaluation of physics acting on a fluid, due to a disturbance in the flow-field. First principle analysis allows for the exact solution of fluid dynamics by considering the forces acting on a fluid element. The basis for CFD forms from three physical considerations:

- The Conservation of Mass

$$\begin{aligned} &Net\,mass\,flow\,out\,of\,Control &=\quad &Time\,Rate\,of\,decrease\,of\,mass\\ &Volume\,through\,surface & &inside\,Control\,Volume \end{aligned}$$

- The Forces acting on a fluid element (The Momentum Equation)

$$F = ma \text{ (Newton's Second Law)}$$

  Net forces acting directly on the mass of the fluid, and surface forces (pressure and shear) acting on a surface of a fluid element.

- The Conservation of Energy

$$\begin{array}{lclcl} Rate\,of\,change\,of & & Net\,flux\,of\,heat & & Rate\,of\,work\,done\,on\\ energy\,inside\,fluid & = & into\,element & + & element\,due\,to\,body\\ element & & & & and\,surface\,forces \end{array}$$

From this basis, the general Navier-Stokes equations can be obtained to describe the state of flow supporting both laminar and turbulent flow conditions. Their exact solution is a time-consuming process and require a high degree of numerical accuracy and stability to adequately capture the high order viscous terms necessary to yield a satisfactory solution.

In this work, it is assumed that a design is sought for smooth bodies that maintain fully attached flow around the body when operating at their design point. Then the viscous effects are most important in a small region near the body profile. In this region, viscous effects may be approximated by describing the flow using boundary layer theory. Outside, an inviscid flow model can be used, and viscous effects assumed negligible.

3.4.1 The Panel Method

In considering irrotational flow where vorticity is zero at every point, the velocity is given by the gradient of the velocity potential ϕ such that

$$V = \nabla\phi. \tag{35}$$

Substituting Equation 35, into the continuity equation for inviscid incompressible flow gives the Laplace Equation as shown in Equation 36

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2}, \quad or \quad \nabla^2\phi = 0 \tag{36}$$

For the flow past a slender body such as an airfoil, a general solution to the Laplace equation can be obtained by adding a distribution of vorticies γ on the airfoil surface, to the potential of the free stream. The solution at a point P due to the distribution of vorticities as illustrated in Figure 12 is given as

$$\phi_P = u_\infty x + v_\infty y + \int_S \gamma\phi_v ds, \tag{37}$$

where ϕ_v is the potential of a unit strength vortex defined as

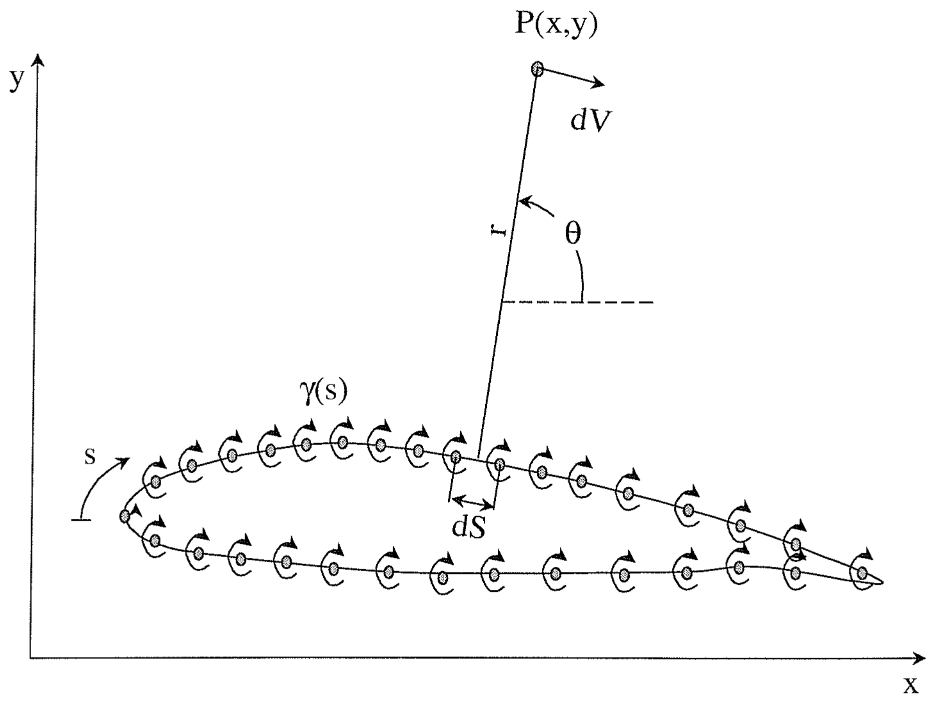

Figure 12: Velocity potential at a point away from a vorticity distribution

$$\phi_v = -\frac{1}{2\pi}\theta \tag{38}$$

with (r, θ) giving the polar coordinates of P relative to $\gamma(s)$.

To satisfy Laplace's equation at every point on the airfoil surface, the Neumann boundary condition gives

$$\frac{\partial \phi}{\partial n} = V_n \tag{39}$$

where $V_n = 0$ would determine a physical boundary with zero flow through it.

3.4.2 Solution of Panel Characteristics

From Laplace's Equation 36, Equation 37 can be written as

$$(u, v)_\infty \cdot n + \int_S \gamma \frac{\partial \phi_v}{\partial n} ds = V_n. \tag{40}$$

By discretising the body surface into N Panels and placing a potential source $\gamma\phi_v$ on each panel, a set of simultaneous equations can be found such that

$$(u, v) \cdot n_i + \sum_{j=1}^{N} \left(\int_{panel} \gamma_i \frac{\partial \phi_{v_i}}{\partial n_i} ds \right) = V_n \tag{41}$$

where $V_n = 0$ for zero normal flow at the surface of the panel.

Treatment of the Kutta condition requires that zero circulation exists at the trailing edge. A simple implementation such as $\gamma_{TE} = \gamma_1 + \gamma_{N+1} = 0$ may lead to unacceptable solutions such as $\gamma_1 = 1e^6$ and $\gamma_{N+1} = -1e^6$. An improved treatment can be applied through the requirement that vorticites must be small, such as $\gamma_1 = \gamma_{N+1} = 0$, which can be achieved through the addition of a constant potential at the wake.

The addition of the Kutta condition to Equation 41, lends to the solution of unknown strengths γ_i. The velocity components at each panel may then be obtained from the tangental components of $(u, v)_i$ such that

$$U_{e_i} = \frac{\partial \phi_{P_i}}{\partial s_i}, \tag{42}$$

which applied to Equation 37, the panel velocities can be obtained from

$$U_e = U_\infty \cdot \mathbf{t} + \sum_{j=1}^{N} \int_{panel} \left(\gamma_i \frac{\partial \phi_{v_i}}{\partial s_i} ds \right). \tag{43}$$

Finally, the coefficient of pressure C_p for each panel is obtained from

$$C_p = 1 - \frac{U_{e_i}^2}{U_\infty^2}. \tag{44}$$

3.5 XFoil - Viscous Coupled Potential Solver

A viscous coupled Panel method XFoil [67] is used extensively throughout this work. Viscous coupling is added to the inviscid irrotational Panel method by including the solution to the integral momentum and kinetic energy momentum equations that describe two-dimensional boundary layer flow. The inclusion of the boundary layer has the effect of effectively modifying the shape of the body as seen by the external flow. To include this effect within the XFoil Panel method solution, the boundary layer displacement thickness δ^* is added to the transpirational velocity condition in Equation 39, such that the normal flow boundary condition becomes:

$$V \cdot n = \frac{d}{dx}\left(U_e \delta^*\right). \tag{45}$$

One approach to applying this two-way coupling, is to solve the panel method equations starting with the Neumann boundary condition $V \cdot n = 0$. Then solve the boundary layer using the panel solution velocity field $\vec{U_e}$ as an initial input condition. The process can then be repeated iteratively replacing the simple Neumann condition with Equation 45 calculated from the boundary layer solution.

It is possible that convergence of the viscous coupling applied through such a method may not always be easily achieved. XFoil adopts a more robust approach by solving the entire non-linear equation set of both Panel method and boundary layer equations simultaneously by a Newton-Raphson method.

A full description of the viscous equations and their solution in XFoil is given in [67]. A brief introduction to their formulation is given below.

3.5.1 Laminar Boundary Layer Approximation

For laminar boundary layer flow, the Navier-Stokes equations can be reduced to derive the Prandtl boundary layer equations

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{46}$$

$$\rho\left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) = -\frac{\partial p}{\partial x} + \mu\frac{\partial^2 u}{\partial y^2} \tag{47}$$

$$\frac{\partial p}{\partial y} = 0 \tag{48}$$

where it is assumed that:

- The boundary layer thickness is very small compared to L for large Reynolds numbers,
- The tangential velocity u is much larger than the normal component v,
- The pressure is essentially constant across the boundary layer in the y direction.

By combining the boundary layer equations, and integrating the resulting expressions to infinity with respect to y, the Von Karmen integral momentum equation is obtained as derived in 49.

$$\frac{d\theta}{dx} + \frac{\theta}{U_e}(2+H)\frac{dU_e}{dx} = \frac{1}{2}C_f \tag{49}$$

where the shape factor $H = \delta^*/\theta$, the displacement thickness δ^*, momentum thickness θ and skin friction coefficient C_f are defined as follows.

$$\delta_* = \int_0^\infty \left(1 - \frac{u}{U_e}\right) dy \tag{50}$$

$$\theta = \int_0^\infty \frac{u}{U_e}\left(1 - \frac{u}{U_e}\right) dy \tag{51}$$

$$C_f = \frac{\tau_w}{1/2 \cdot \rho U_e^2}, \; where: \; \tau_w = \mu \left.\frac{\partial u}{\partial y}\right|_{y=0} \tag{52}$$

By multiplying the momentum integral equation by u and integrating, the kinetic energy integral equation is obtained:

$$\frac{d\theta^*}{dx} + 3\frac{\theta^*}{U_e}\frac{dU_e}{dx} = 2C_D, \tag{53}$$

where the kinetic energy thickness θ^*, and the dissipation coefficient C_D are defined as

$$\theta_* = \int_0^\infty \left(1 - \left(\frac{u}{U_e}\right)^2\right) \frac{u}{U_e} dy \tag{54}$$

$$C_D = \frac{1}{\rho U_e^3}\int_0^\infty \mu \frac{\partial^2 u}{\partial y^2} dy \tag{55}$$

The system of equations 49 and 53 contains too many unknowns, and therefore must be complemented by other equations. XFoil uses the semi-empirical relations from the Falkner-Skan velocity profile family to close the system, which are solved using a backward Euler discretisation. A full description of the solution used by XFoil to the Laminar boundary equations is discussed by Drela [67].

3.5.2 Implementation of the Turbulent Boundary Equations

In turbulent flow conditions, fluid motion becomes unsteady. In such a case, following the fluid motion in detail becomes impractical, and therefore a description of the average motion of the flow is substituted.

If we define the time average of any flow quantity by:

$$\bar{a}(x,y) = \lim_{T \longrightarrow \infty} \frac{1}{T} \int_{t_0}^{t_0+T} a(x,y,t)dt \tag{56}$$

where

$$a(x,y,t) = \bar{a}(x,y) + a'(x,y,t)$$

such that a' represents the fluctuations from $(a - \bar{a})$.

By time-averaging and comparing the order of magnitude of terms of the steady incompressible Navier-Stokes equations, the following turbulent boundary layer equations can be derived:

$$\frac{\partial \bar{u}}{\partial x} + \frac{\partial \bar{v}}{\partial y} = 0 \tag{57}$$

$$\rho \left(\bar{u}\frac{\partial \bar{u}}{\partial x} + \bar{v}\frac{\partial \bar{u}}{\partial y} \right) = -\frac{\partial \bar{p}}{\partial x} + \frac{\partial}{\partial y} \left(\mu \frac{\partial \bar{u}}{\partial y} - \rho\overline{u'v'} \right) \tag{58}$$

$$\frac{\partial \bar{p}}{\partial y} = \frac{\partial}{\partial y} \left(\mu \frac{\partial \bar{v}}{\partial y} - \rho\overline{v'^2} \right) \tag{59}$$

The Von Karmen integral equations can be derived from the turbulent boundary layer equations. The main differences to the laminar cases include the inclusion of a Reynolds stress term $-\rho\overline{u'v'}$, and the integral quantities δ^*, θ, H and C_f are also expressed as time-averaged velocities. As in the laminar case, solution of the Von Karmen integral equations requires the inclusion of additional equation relationships to complete the system. XFoil incorporates relationships derived from the Falkner-Skan profile family to complete the solution which is detailed by Drela in [67]. Turbulent separation is predicted to occur when H approaches 3.3.

3.5.3 Location of Transition

During the growth of the laminar boundary layer, disturbances can make the flow layer become unstable. Eventually the disturbances cause the boundary layer to become turbulent, which is known as transition. The accurate prediction of the transition region is important in drag estimation as it defines regions of laminar flow where skin-friction is low, and turbulent regions where skin-friction drag increases rapidly.

XFoil adopts a procedure for transition prediction known as the e^9 method, which computes the maximum spacial amplification ratio formed from the growth of wave like disturbances in shear layers. Transition is most likely to occur when the amplitude has grown by more than a factor of e^9, although the ratio can vary between e^7 to e^{11} owing to factors such as surface roughness and free stream turbulence.

From the Falker-Skan profile family, the spatial amplification curves based on the Orr-Sommerfield equation can be related to the local boundary layer parameters [67]. The amplification curve envelopes are approximated such that

$$\tilde{n} = \frac{d\tilde{n}}{dRe_\theta}(H)\left[Re_\theta - Re_{\theta_0}\right], \tag{60}$$

where $\tilde{n}$ is the logarithm of the maximum amplification ratio, $Re_\theta = ReU_e\theta$, and the critical Reynolds number Re_{θ_0} are expressed by the following empirical formulas:

$$\begin{aligned}
\frac{d\tilde{n}}{dRe_\theta} &= 0.01\left(\left[2.4H - 3.7 + 2.5\tanh\left(1.5H - 4.65\right)\right]^2 + 0.25\right)^{\frac{1}{2}} \\
\log_{10}\left(Re_{\theta_0}\right) &= \left(\frac{1.415}{H-1} - 0.489\right)\cdot\tanh\left(\frac{20}{H-1} - 12.9\right) + \frac{3.295}{H-1} + 0.44
\end{aligned}$$

The amplification rate is obtained by integrating the amplification rate downstream from the instability point $x_{critical}$ where $Re_\theta = Re_{\theta_0}$, giving:

$$\tilde{n}(x) = \int_{x_{critical}}^{x} \frac{d\tilde{n}}{dx} dx. \tag{61}$$

The conversion from Re_θ to spatial x is accomplished by:

$$\frac{d\tilde{n}}{dx} = \frac{d\tilde{n}}{dRe_\theta}\frac{dRe_\theta}{dx} = \frac{d\tilde{n}}{dRe_\theta}\frac{1}{2}\left(\frac{x}{U_e}\frac{dU_e}{dx} + 1\right)\frac{\rho U_e \theta^2}{\mu x}\frac{1}{\theta} \tag{62}$$

The amplification rate is expressed in terms of H and θ by:

$$\frac{d\tilde{n}}{dx}(H,\theta) = \frac{d\tilde{n}}{dRe_\theta}(H) \cdot \frac{m(H)+1}{2} p(H) \frac{1}{\theta} \tag{63}$$

and

$$\begin{aligned} \frac{\rho U_e \theta^2}{\mu x} &= p(H) = \quad (6.54H - 14.07)/H^2 \\ \frac{x}{U_e}\frac{dU_e}{dx} &= m(H) = \quad \left(0.058\frac{(H-4)^2}{H-1} - 0.068\right)\frac{1}{p(H)} \end{aligned} \tag{64}$$

3.5.4 Implementation of Boundary Layer Calculation and Prediction of Drag

The numerical approximation for the Laminar boundary layer is first calculated by growing the solution from the stagnation point near the leading edge of the airfoil. Boundary layer transition is then added to the solution. Since no method exists for describing the transition process, a fictious transition point is used to define a point where the boundary layer instantaneously changes from a laminar boundary layer to a turbulent one. This transition point is located from the laminar boundary layer where $\tilde{n}(x) = 9$. In XFoil, an interval $(i, i+1)$ is first found such that $\tilde{n}(x_i) < 9$ and $\tilde{n}(x_{i+1}) > 9$, and then linear interpolation is used to find the exact point, re-evaluating the brackets at each iteration. The turbulent boundary layer calculation is then applied from the transition point starting with initial values taken from the end of the boundary layer calculation.

In two dimensions, the components of total drag reduce to that of profile drag which can be calculated from the loss of momentum across the wake. The Squire-Young formula is used to calculate profile drag at the last computational point in the wake.

$$C_D = \frac{D}{1/2\rho U_\infty^2} = s\theta_\infty = 2\theta_{TE}\left(U_e\right)^{\frac{H_{TE}+5}{2}}. \tag{65}$$

3.5.5 Parameterisation of XFoil for Optimisation

The setting of XFoil parameters should appropriately reflect the solvers ability to capture aerodynamic sensitivity to a wide range of designs. The accurate prediction of boundary layer transition will have a critical impact on drag calculation. To help define boundary layer parameters, the drag polar of the NACA0012 airfoil has been reconstructed and presented in Figure 13. In XFoil, the transition point is given as the lesser of $n(x) = n_{critical}$ and $x = x_{trip}$ where $e^{n_{critical}}$ is the maximum amplification ratio for disturbance growth described in Section 3.5.3, and x_{trip} defines a point where the boundary layer becomes turbulent if not already so. The parameter $n_{critical}$ is mainly determined by surface roughness and external parameters such as temperature and free stream turbulence. These factors are considered beyond the scope of this study and therefore the value of 9 is used. The values for x_{trip} for the upper and lower surface were set at $x/c = 0.3$ for the upper surface where the flow is expected to become turbulent early in normal operation, and $x/c = 0.5$ for the lower surface where the flow is expected to trip later owing to the reduced curvature normally found in the first part of the section.

The drag polar produced by XFoil for $Re = 3e^6$ is compared in Figure 13 to experimental results obtained from [68]. The results obtained by introducing boundary layer tripping, compares more favourably, than the results obtained where the boundary layer is allowed to trip freely as determined by $n(x) = e^9$. This difference is expected to be largely attributed to the manual tripping of the boundary layer on the experimental model at $x/c = 0.3$. The pressure coefficient distribution for the upper surface at zero angle of attack is compared with experimental results in Figure 14.

Overall, a reasonable capture of the aerodynamic characteristics important in this

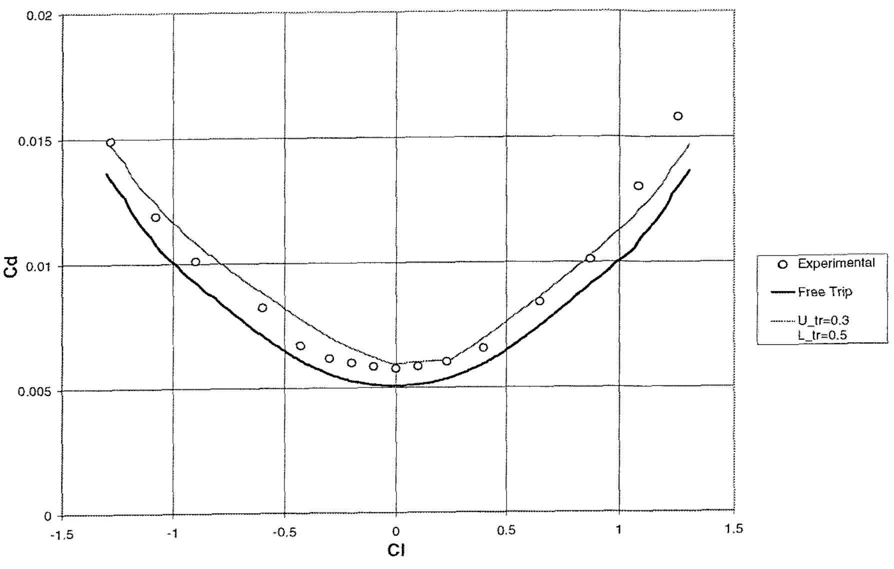

Figure 13: XFoil Drag Polar Reconstruction for a NACA0012 Airfoil

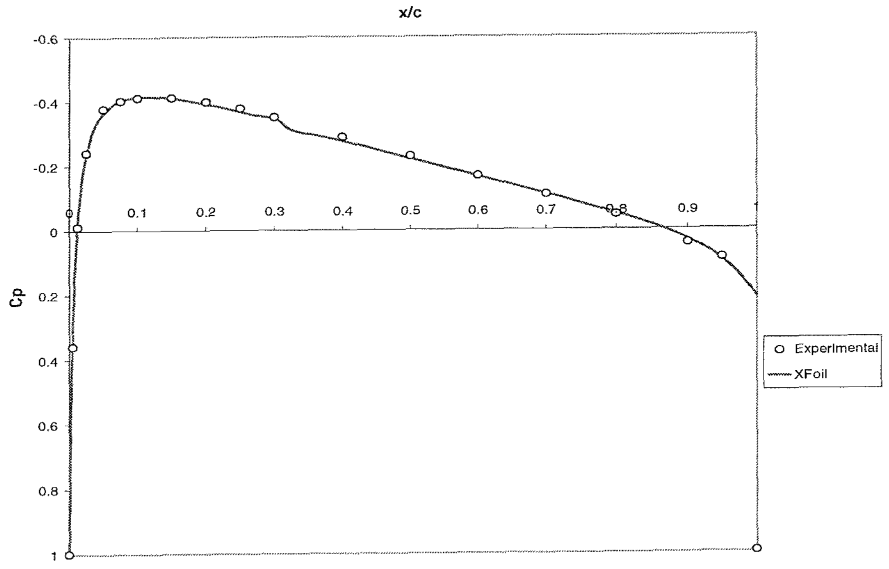

Figure 14: XFoil Pressure Distribution Reconstruction for a NACA0012 Airfoil at $C_L = 0.0$

work can be made by XFoil, which offers a cheaper alternative to the more expensive viscous solutions of the Navier-Stokes equations.

3.6 Unstructured Cell Vertex Euler Solver

The Euler equations describing the conservation laws of mass, momentum, and energy, can be written in vector form as:

$$\frac{\partial U}{\partial t} + \vec{\nabla} \cdot \vec{F} = 0 \tag{66}$$

where $\vec{F} = (f, g, h)$, and

$$\mathbf{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{bmatrix} \quad f = \begin{bmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ \rho uw \\ \rho uH \end{bmatrix} \quad g = \begin{bmatrix} \rho v \\ \rho vu \\ \rho v^2 + P \\ \rho vw \\ \rho vH \end{bmatrix} \quad h = \begin{bmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + P \\ \rho wH \end{bmatrix}$$

E and H are the total energy and stagnation enthalpy per unit mass respectively. Equation 66 can be integrated over an arbitrary finite volume Ω to construct the integral form of the equation.

Using Gauss' Divergence Theorem this can be expressed as

$$\frac{\partial}{\partial t} \int_{\Omega} U d\Omega = - \oint_{\partial\Omega} (f n_x + g n_y + h n_z) dS \tag{67}$$

where $\partial\Omega$ represents the contour around the volume Ω and S represents the surface of the volume. An average change of the conserved variables, denoted by $\overline{U}$can be expressed for a discrete finite volume as

$$\left(\frac{\partial \bar{U}}{\partial t}\right)_{\Omega} = \frac{-1}{V_{\Omega}} \sum_{k=1}^{n} (f S_x + g S_y + h S_z) dS \tag{68}$$

where the summation is over all the faces of the discrete volume, S_x, S_y and S_z represent the projected areas of these faces, and V_{Ω} is the volume. Equation 68 can be used over any discrete volume thus the method can be applied on any grid topology.

A Cell Vertex scheme is used to simplify the boundary condition implementation. Additional volumes or 'Control Volumes' are constructed over which the equations are integrated. The flux values on the faces are calculated using Roe's upwind scheme [69], in which an approximation to the Riemann problem is sought on the control volume faces.

The numerical flux, equivalent to the terms inside the right hand integral of Equation 67, can be expressed as

$$f^*_{i+1/2} = \frac{1}{2}\left[f(U^R) + f(U^L) - |A_{Roe}|\,(U^R - U^L)\right] \tag{69}$$

Where $|A_{Roe}|$ is the flux Jacobean, evaluated using Roe's fluid state [69].

In the implementation used, Equation 68 is used to calculate U_i^{n+1} from U_i^n for each volume by calculating the numerical flux given by 69 on each face and then integrating the result with respect to t via Runge Kutta numerical integration. A detailed description of the implementation of the solver is given by Rycroft [70].

3.7 Summary

Two flow solvers have been described in this Chapter, XFoil and an unstructured $3D$ Euler solver. XFoil has been found to suitably reflect the incompressible flow characteristics of a NACA0012 section. It will be used for the optimisation study of the ducted thruster foil section, due to its rapid evaluation of the flow solution. The Euler solver will be used to measure the scalability and performance of a distributed PC network for aerodynamic calculation.

Chapter 4

The Formation of Species and Optima Finding with Genetic Algorithms

For this work, a robust Genetic Algorithm capable of discovering novel design features amidst multi optima and deceiving search landscapes is required. Without this basic capability, research into shape adaptivity may become influenced by the noise of unnecessary disruption due to local optima traps, population dominance by a single species or member, or through other biases that may hamper the convergence and adaptation ability of the algorithm. This chapter is concerned with the construction of such an algorithm.

4.1 The Canonical Genetic Algorithms

The evolutionary cycle that encapsulates the Genetic Algorithm discussed in section 2.2.2 has been implemented using C++. The Canonical Genetic Algorithm which features binary string chromosome structures, has been used as a starting point model due to its success in previous implementations [2][36], and the large amount of research that has been carried out around it. Other implementations of the GA may also be used for this work including Real encoding as described by Eshelman and Schaffer

[71], as well as the more advanced Breeder GA [72], and Messy GA [16]. However these are generally harder to implement on real problems, and may not necessarily help any more in achieving the goals of this research.

A description of the Object Orientated implementation of the GA used in this work is given in Appendix B. This section discusses the key algorithms and operators used, and their effect on its performance when applied to a simple but difficult optimisation problem, 'The Bump Problem'.

4.1.1 The Bump Problem

The bump problem, introduced by Keane [73], is a difficult problem for most optimisation methods to deal with. Good local optima are found in small islands or bumps, surrounded by vast areas of poor solutions. Such a landscape is difficult to explore as the global optimum which lies on a constraint boundary cannot be found by simply following a path improving fitness or good solutions. It is assumed that the periodic formation of the peaks will not assist the search process. The orthogonal structure of the bumps may assist operations such as genetic crossover, however the occurrence of such an event it considered to be extremely small, and its advantage to the operator ignored. The bump problem is ideal for testing search algorithms (Schoenauer and Michalaewicz, 1996 [74];Michalewicz *etal.*, 1998 [75]; and Keane, 1995 [73]) including those destined for aerodynamic design where a slight perturbation in the design parameters can lead to disastrous results.

The problem is defined as:

$$Maximise \frac{abs\left(\sum_{i=1}^{n} cos^4(x_i) - 2\prod_{i=1}^{n} cos^2(x_i)\right)}{\sqrt{\sum_{i=1}^{n} ix_i^2}} \tag{70}$$

for

$$0 < x_i < 10 \quad i = 1, ..., n \tag{71}$$

subject to

$$\prod_{i=1}^{n} x_i > 0.75 \quad and \quad \sum_{i=1}^{n} x_i < 15n/2 \tag{72}$$

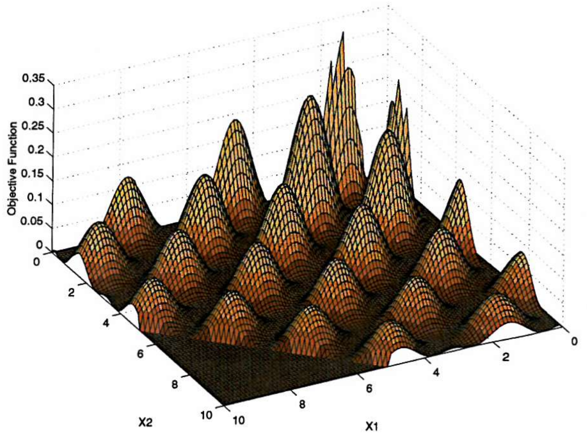

Figure 15: 2D Bump Problem

starting from $x_i = 5$, $i = 1, ..., n$. Figure 15 shows the 3D plot of the bump problem for $n = 2$. For $n > 2$ this problem becomes more demanding, with families of peaks occurring within a complex constraint surface.

4.1.2 GA initialisation, establishing generation zero

The initial population is usually setup by random generation of member chromosomes, which are then evaluated and stored until the initial population is complete. The length, type and structure of the chromosome string depends on the number of parameters to be discretised, the size of the individual parameter space, and the representation scheme used.

The structure of the chromosome is defined by genes representing the individual parameters. As with any optimisation process the number of parameters used will increase the number of objective functions to be evaluated.

Binary Encoding

A simple but extremely effective genetic coding is achieved using finite length binary strings to encode the search variables. A variable can be represented by enforcing limits on the variable range such that $min_i \leq x_i \leq max_i$ where x_i is the parameter to be searched. A gene string is encoded for each variable by subdividing the parameter by the number of binary bits to be used.

To ensure that the encoding can survive mutation and crossover to the best of its ability, Grey Coding has been used here to reduce the binary 'Hamming Distance' between genes. Hamming Distance is the number of different positions between two binary vectors. For example "01" would represent the integer value of 1, whereas flipping the left bit to "11" would now yield a value of 4. Using Grey coding, the vector "11" would yield a value of 2. Davis [76] found that the introduction of Grey Coding to the chromosome encoding helped members survive after undergoing mutation operations where binary code bits were randomly flipped. The Grey Code implementation given in [19] was used in this work.

The chromosome is made by concatenating all the gene strings together. Each bit of the combined chromosome string is known as an allele, and its position is called the locus. Genetic operators can be performed on this chromosome string, before it is decoded back into its real number components for evaluation.

Population Sizing

Population size is one of the most important parameters to be considered on tailoring a GA. Despite the depth of literature found evaluating various strategies for determining the appropriate population size, there are still no firm guidelines of how large a population should be used for a particular problem. If the population is too small, progress can be hindered by the lack of schema sampling available within its members, and increased stochastic effects are required to alleviate this problem. Too large a population and computational cost can become a concern.

Convergence time is dependent on population size. Goldberg[77] showed that typical GAs converge on $\mathbf{O}(Plog\mathbf{P})$ generations, where $\mathbf{P}$ is the population size. Since convergence is used to indicate the low rate in which the populations fitness progresses at each generation (stops improving), it would be suggested that large populations should be used to avoid premature convergence.

Initialisation

The initial starting point of the GA is not critical as in other optimisation methods. It is still important to ensure that the initial population is spread over a diverse area of the search space.

The three main methods used to initialise a population are random generation, deviate and injection. In random initialisation, chromosomes of each population member are created bit by bit via random number generation. In its simplest form, a binary chromosome is created by generating random numbers of 0 or 1 for each bit until the chromosome is complete, without any knowledge of individual gene representation. In deviate initialisation, a given set of chromosomes are perturbed by a random amount. This is done by real/integer number perturbation on a gene by gene bases. The third initialisation method, injection initialisation, reads the chromosome representation from file either directly in their binary string form or from their actual parameters which are converted to binary. All three schemes are included in the GA. The deviate method is necessary where the feasible solution space is sparse and formulation of a diverse population with a relatively high degree of feasibility difficult. Injection allows for restarts and structured experiments to be performed.

On creation of all the population chromosomes, the population is evaluated establishing generation zero. In search areas containing a high degree of infeasibility, it may be necessary to continue initialising new members until a predefined proportion of feasible members is reached in generation zero.

Evaluation of the objective values is then performed to assign each Genome with its raw fitness measure for the selection process.

Constraint Handling

In order to solve a constrained optimisation problem, it is important to ensure that any optima found satisfy all constraints. A variety of constraint handling methods can be used with the various optimisation techniques and a general discussion of the approaches that can be used is given by Siddal [18]. Penalty functions are a simple approach to handling the constraint problem commonly adopted to work with evolutionary algorithms due to their simplicity, robustness and the fact that they do not require knowledge of sensitivity. Penalty function methods, work by combining the objective function with some penalty function that penalises the objective value when a constraint is violated.

The simplest form of a penalty function is given in Equation 73.

$$U_P(x) = U(x) + 10^{20} \sum_{j=1}^{M} |c_j^S| + 10^{20} \sum_{j=1}^{L} |c_j^V| \tag{73}$$

The expression represents a distorted objective function which tends to force the search towards feasible areas. If the search either begins in the infeasible region, or wanders into it, it is urged towards the feasible region by the distortion of the surface. In this work, Fiacco and McCormick Penalty functions are used:

$$U_P(x) = U(x) + \frac{1}{r} \sum_{j=1}^{L} c_i^V(x)^2 + r^2 \sum_{j=1}^{M} \frac{1}{c_j^S(x)} \tag{74}$$

where $r = [0, 1)$ is a penalty relaxation factor used to control the intensity of the applied penalty.

Stochastic search methods may find it difficult to satisfy an equality constraint, and therefore in this work only inequality constraints are considered (equality constraints can easily be treated as inequality constraints). The constraint functions are defined as:

$$c_j = C_T - C(x) \tag{75}$$

where $C(x)$ is the value of the constraint and C_T is the constraint target value.

This method contains mixed penalty functions in both the interior (feasible) regions which are a component of the satisfied constraints, and an exterior (infeasible) component which is a function of the violated constraints. The internal term increases in size as the constraint boundary is approached, trying to prevent the solution from crossing the boundary region. As the solution matures and becomes less erratic, the penalty given by the interior and exterior terms close to or at the constraint boundary is reduced sharply, thus allowing the solution to approach the boundary if required. The penalty reduction factor r is typically reduced at a rate of 0.04 per generation.

4.1.3 Parental Selection

The objective of the selection process is to ensure that only good quality schema participate in the reproduction stage. The selection method must therefore be biased towards members of above average fitness, such that high quality schema are likely to propagate into future generations. Since the selection stage determines the type of schema sampled by the Genetic Algorithm it is often considered as the most important phase in the evolution process. In order that a judgement can be made between good and poor candidates, some measure of the solution fitness must be made. The members fitness is a relative term which indicates the individuals standing among its contemporaries. If $\vec{\psi}$ is the set of input parameters and $\vec{\Phi} = E(\vec{\psi})$ is the vector of objective variables returned by the objective function, then fitness F is given by

$$\vec{F}(\vec{\psi}) = f(E(\vec{\psi}), t) = f(\vec{\Phi}, t), \tag{76}$$

where t is the generation index of the algorithm. The success of a GA lies in exploiting existing information from its members while exploring new areas of the search space, by sampling schema from a diverse population. Selection therefore plays a key role in this process. If the selection is too elitist, and heavily biased towards selecting the best members of each population, then this intensive selection pressure can lead to a significant reduction in the phenotypical diversity of the reproduction pool [78], thus reducing the exploration advantage of the GA. The result can lead to premature convergence of the evolution process. Problems of population take-over where the population becomes swamped with elitist members, are frequently reported with such applications.

To examine such key performance characteristics, several terms were defined by Baker [79] and Back [80] as summarised below.

Selection Pressure: the probability of the best individual being selected compared to the average probability of selection of all the individuals.

Bias: the difference between an individual's normalised fitness and its expected probability of reproduction.

Spread: the range of possible values for the number of offspring of an individual.

Loss of Diversity: the proportion of individuals of a population that are not selected during the selection phase.

Selection Intensity: the expected average fitness value of the population after applying the selection method to the normalised Gaussian distribution.

Selection Variance: the expected variance of the fitness distribution of the population after applying the selection method to the normalised Gaussian distribution.

Fitness Assignment

The first stage of a selection process is to award each individual of the current population pool, a weighting based on its relative performance amongst the other competing members. The weighting which is based on the notion of member fitness, will give the individual an appropriate probability of being selected in the reproductive selection process.

Fitness assignment essentially provides a measure of the performance of an individual's relative strength amongst the other candidates, against the objective of the evolution problem. The original or raw fitness measure of a member is usually determined by the objective function. Although some selection methods can work with this raw assignment, it is sometimes more useful to re-scale this performance measure so that the differences between members becomes more subtly defined.

Linear scaling is simple to implement, where member fitness is scaled using a linear function dependent on a predetermined value.

$$\Phi_{i,t} = \frac{1}{1 + f_{max,t} - f_{i,t}} \tag{77}$$

where $\Phi_{i,t}$ is the positive fitness assigned to an individual i in generation t, $f_{max,t}$ is the maximum member's raw fitness assigned by the objective function and $f_{i,t}$ is the objective fitness for member i in generation t. The resultant fitness using this simple linear assignment lies in the range of $(0, 1]$.

This method, can however lead to intense local selective pressure on the elitist proportion of the population, reducing the phenotypical diversity of future generations. Exponential and power based proportionate scaling methods, can be introduced to alleviate the pressure on local selection. However phenotypical distances are still the main measure on which the method is based and are not necessarily conducive to exploration.

Ranking methods (i.e. first, second, third, ...) [81] ignore phenotypical distances where the measurement of fitness is based on the members position relative to the rest of the population. If two or more individuals have equal standing, they are assigned an equal averaged rank. Three types of ranking scales are generally adopted, linear, power and exponential, and the way these are used generally depends on how much selection pressure is to be applied. The linear ranking scheme used was assigned such as

$$\Phi_{i,t} = \frac{\alpha_{rank} + \left[rank(i)/\left(P(t) - 1\right)\right]\left(\beta_{rank} - \alpha_{rank}\right)}{P(t)} \tag{78}$$

where α_{rank} is the expected number of offspring to be allocated to the worst member and β_{rank} is an adjustable parameter that defines the proportion to be allocated to the best ranked member such that $\alpha_{rank} = 2 - \beta_{rank}$, and $1 \leq \beta_{rank} \leq 2$. $P(t)$ is the population size of generation t.

If this fitness assignment yields excessive convergence times, then a higher level of selection pressure can be applied using an exponential scale. For the exponential scheme, fitness from best to worst are assigned as $1, s^j, s^{j+1}, ...,$, where $s < 1.0$ (such as $s = \frac{P-1}{P}$) and j is a positive integer. Both s and j can be varied to apply the appropriate level of selection pressure.

Fitness Selection

The objective of fitness selection is to create the pool of new parents from the current population, by associating higher chances of selection with members of higher assigned fitness.

"Roulette Wheel" selection [11] is the most basic mechanism used by many GAs due to its simplicity. This method is analogous to replacing the numbers usually given on a roulette wheel with differently spaced pockets for each member. The size of each spacing is determined by that member's cumulative probability q_i which is given by:

$$\begin{array}{c} q_i = \sum_{j=1}^{i} p_j, \quad where \\ p_j = \frac{\Phi_{j,t}}{F(t)}, \quad F(t) = \sum_{i=1}^{popsize(t)} \Phi_{i,t} \end{array} \tag{79}$$

where $\Phi_{i,t}$ is given by Equation 78. The roulette wheel is then spun by randomly dialing a number $(0, 1]$, and the member with the cumulative probability that spans the dialled number, is selected to participate in the next generation. If the selection scheme allows the replacement of individuals drawn from the candidate pool, then the wheel is simply spun *popsize* times. If a no-replacement scheme is desired, then the cumulative probability q_i must be recalculated each time a member is selected.

Although this scheme is simple and versatile, compared to the different fitness assignment schemes available, it is prone to generating numerical noise; i.e. the variance of the probability density function for selecting a member of given fitness is relatively large. Numerical noise generated by the roulette wheel and reproduction operators, tends to obscure the signal difference between good and bad schemata.

The Stochastic Universal Sampling (SUS) scheme was proposed by Baker [79] to eliminate the numerical noise found in the Roulette Selection. The implementation of this scheme is illustrated in figure 16.

This scheme works in the same way as roulette selection, but only requires a single random number to define the location of the first selection point. Then n-1 points are chosen from this first point at constant intervals. Because only one number is drawn from a random process, the appearance of each member in the reproduction pool is proportional to its expected probability of selection. A disadvantage of this scheme is that similar members with poor fitness are severely screened from the selection process, eliminating a substantial part of the stochastic behaviour of the selection role.

Another simple selection scheme that has become popular amongst researchers is Tournament Selection [82][83]. In this scheme a tournament is held among a set of

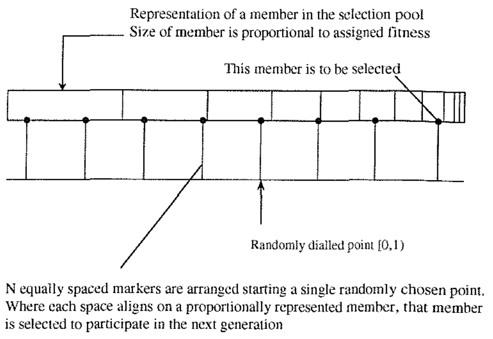
Representation of a member in the selection pool
Size of member is proportional to assigned fitness
This member is to be selected
Randomly dialled point [0,1)
N equally spaced markers are arranged starting a single randomly chosen point. Where each space aligns on a proportionally represented member, that member is selected to participate in the next generation

Figure 16: Stochastic Universal Sampling Scheme

individuals. The set is chosen at random from the selection pool, and the tournament winner is selected to become a parent, and the process repeated until the parent pool is filled. Like the Roulette Wheel and SUS, this method can be used both with or without replacement of candidates in the selection pool. The size of the sub-pool is defined by the tournament size *Tour*, taking values ranging from 2 to *popsize*. Numerical noise is reduced as roulette type routines are no longer required, while randomness is maintained through the selection of the member set participating in the tournament. Tournament victory is usually determined using any of the fitness assignment methods mentioned above.

Elitism

"Elitism", was first introduced by Kenneth De Jong(1975), as an addition to many selection methods forcing the GA to retain a number of best candidates at each generation. Such individuals can be lost if they are not selected to reproduce, or are destroyed through the reproduction process. A parameter p_{best} is used in this implementation to define the proportion of best individuals that are to be preserved to the next generation without undergoing selection or reproduction processes.

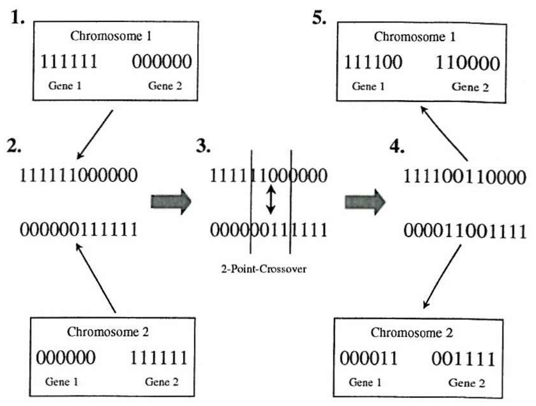

Figure 17: Two Point Crossover Scheme

4.1.4 Reproduction

In the reproduction process, two individuals share genetic information to produce offspring that contain characteristics of their parents. In the GA implemented here, two of the basic operators are used, crossover and mutation.

Crossover

Traditional binary crossover takes the two parents, cuts their chromosome strings at randomly chosen locations, which are then swapped over from one chromosome string to the other as illustrated in Figure 17.

The offspring thus inherit some genes from each parent with a small amount of genetic disruption. Two common parameters, crossover rate and the number of crossover points are usually used to define the type of crossover that takes place. Crossover rate defines the probability of crossover occurring during the reproduction phase (typically between 0.6 and 1.0). If the crossover rate is 1.0 then all offspring will contain genetic material from both parents, if a rate of 0.6 is used, then approximately 60% of individuals undergo crossover. The number of crossover points determines the number of times the parent chromosome is cut and swapped with a second parent chromosome. Generally single point crossover is used, but other common rates include two point crossover as indicated in figure 17, and uniform crossover [14] where every other allele is exchanged between chromosomes.

Mutation

Mutation is applied to each offspring (after crossover), randomly altering each gene with a small probability (typically 0.001 to 0.01). The role of mutation is to provide a small amount of random search, and helps to ensure that no point in the search space has zero probability of being explored.

Inversion

Inversion is applied in a similar manner to mutation, except the result is a slight reordering of allele bits, such that **110** would become **011**. The implementation used in this work will allow three bits of a chromosome string to undergo inversion at a probability p_{inv}.

4.1.5 Death

The occurrence of death to each member is generational. That is, at each generation, a new population is created from the selected parents. New children are created by taking two parents at a time and performing crossover. If crossover is not performed, then the parents are copied into the new population as new members. Mutation and inversion is performed on all members of the new population. Finally elitism is applied to ensure the previously best member is represented in the new population. The previous population is completely destroyed.

4.1.6 Implementation

The GA and evolutionary operations described above, has been implemented using C++ as described in Appendix B. The basic logical flow process describing the artificial evolution process is illustrated in Figure 18.

```
1. Initialise template DNA Structure.
2. Initialise $Parent$ Genome Population with template DNA.
3. Perturbate Parent DNA.
4. Evaluate Population $rawFitness$.
5. Save Generation 0 Stats.
6. $NChild = NPopsize - NElite$
7. For $i = 0$ to $i = NGens - 1$
    (a) Sort Parent Population by $rawFitness$.
    (b) Rank Population by $rawFitness$, $\Rightarrow$ $scaledFitness$.
    (c) Calculate $\sum scaledFitness$
    (d) Calculate array of accumulative $scaledFitness/(\sum scaledFitness)$.
    (e) Select $NChild$ parents via SUS Selection scheme and copy to $ParentPool$.
    (f) While $Size(ParentPool) > 0$
        i. Remove random $Genome_A$ and $Genome_B$.
        ii. For $j = 0$ to $j = NCross - 1$
            A. Crossover $Genome_A$ with $Genome_B$.
        iii. Apply mutation to $Genome_A$ and $Genome_B$.
        iv. Apply inversion to $Genome_A$ and $Genome_B$.
        v. Add $Genome_A$ and $Genome_B$ to $ChildPool$.
    (g) Evaluate $ChildPool$.
    (h) Copy $NElite$ members from $ParentPool$ to $ChildPool$.
    (i) Save Generation Stats for $Generation\ i$.
    (j) Copy $ChildPool$ to $Parents$.
8. Save Stats to file.
9. End.
```

Figure 18: Implementation of the Canonical Genetic Algorithm

4.2 Performance Evaluation of the Canonical Genetic Algorithm

Proportionate, linear ranking, exponential ranking and tournament selection methods have been explored for their suitability for use with difficult objective functions such as the Bump problem. The probability of mutation occurring during reproduction was maintained at 2%, and a crossover probability of 0.6 was investigated. The bump problem was used with $n = 2, 5, 20$ scenarios and the tolerance of the binary encoding set to 20bits per gene. The population size was set to 50 members for $n = 2$ and 300 members for $n = 5$ and $n = 20$. Elitism was used to ensure that the best member of each generation survived to the next one. The penalty function given in Equation 74 was used to steer the solution away from infeasible areas, with the reduction factor r

initiated at 1.0, and reduced by an amount of 0.04 per generation. The results matrix for each GA configuration is shown in Table 4 for $n = 2$ case, Table 5 for $n = 5$, and Table 6 for $n = 20$ bump problem.

GA Scheme	p_{cross}	1	2	3	4	5	Avg
Linear Ranking	0.6	0.18	0.21	0.24	0.19	0.24	0.21
Linear Ranking	0.85	0.15	0.15	0.18	0.13	0.13	0.15
Exponential Ranking	0.6	0.12	0.11	0.10	0.12	0.14	0.12
Exponential Ranking	0.85	0.11	0.12	0.15	0.09	0.11	0.12
Tournament Selection	0.6	0.21	0.15	0.18	0.22	0.16	0.18
Tournament Selection	0.85	0.18	0.20	0.18	0.19	0.15	0.18

Table 4: Performance of Various Selection Schemes on 2D Bump problem (5 Samples)

GA Scheme	1	2	3	4	5	Avg
Linear Ranking	0.42	0.38	0.26	0.34	0.41	0.36
Exponential Ranking	0.24	0.32	0.36	0.28	0.40	0.32
Tournament Selection	0.32	0.36	0.27	0.33	0.38	0.33

Table 5: Performance of Various Selection Schemes on 5D Bump problem (5 Samples)

GA Scheme	1	2	3	4	5	Avg
Linear Ranking	0.54	0.46	0.52	0.37	0.41	0.46
Exponential Ranking	0.34	0.30	0.42	0.28	0.32	0.33
Tournament Selection	0.42	0.25	0.43	0.38	0.34	0.36

Table 6: Performance of Various Selection Schemes on 20D Bump problem(5 Samples)

The selection pressure offered by Exponential ranking and Tournament selection were found to be far too excessive for the bump problem when starting from a fixed initialisation point. The solutions converged extremely quickly for these configurations, and population takeover (where clones of the elite members dominate in the population) was observed to occur after just 20 generations for the $n = 20$ bump problem. Linear ranking with a selection intensity of 1.6 was able to prolong convergence for at least 40 generations for the same case, while consistently finding superior results. Figure 19 shows a resultant population produced by Linear ranking with $p_{cross} = 0.85$. This result shows population convergence on a large local optimum in a path between the starting point at $x_1 = 5, x_2 = 5$, and the optimum. Of course if a totally random

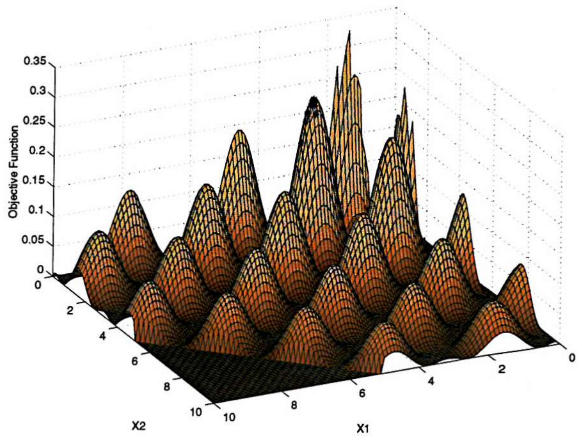

Figure 19: Typical Population Convergence (popsize 50, nGens 40)

starting generation was used instead of perturbing a point, we would expect some members to find the optimum some of the time. However from a fixed starting point, only six solutions found the optimum out of 50 trials.

For populations created through pure random generation as opposed to some perturbation around some starting chromosome string, a slight improvement in convergence can be gained as shown in Figure 20 for a series of five trials. For this solution, higher mutation and inversion rates were applied to a faster converging Tournament selection scheme. Without the complete random generation, premature convergence was found to occur nine out of ten solutions.

The results shown here could possibly be improved slightly through further parameter tuning, however the results from the different selection and crossover schemes illustrate the limit in improvement that is available particularly at higher dimensions where fitness values in excess of 0.7 exist for the 20D Bump problem [73]. One aspect that is absent from the present GA implementation, is that of population diversity control, where different species co-exist but share different resources.

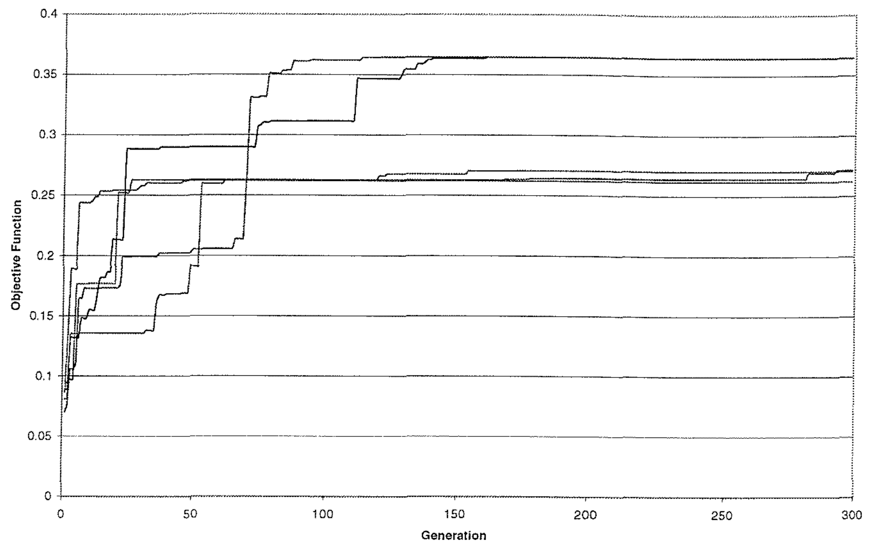

Figure 20: Convergence of the Bump Problem with Tournament Selection (popSize 50)

4.3 Maintaining Population Diversity

4.3.1 The Island Genetic Algorithm

The Parallel Genetic Algorithm (PGA) was first proposed by Tanese [84][85], as a way of efficiently parallelising the canonical genetic algorithm on hypercube computers. The implementation divides the population into several sub-populations, one per processor. Each subpopulation is evolved as a separate canonical genetic algorithm. At certain generational intervals, inter-processor communication occurs, where members from each subpopulation are allowed to move across to another subpopulation. This process is known as the migration phase. Migrated members are either added to the existing population, or replace existing members of its new subpopulation by some replacement process. Once migration has occurred, the evolution process is resumed, until the next migration phase.

Tanese found that not only did this implementation produce near linear speedup in the evaluation of candidates through parallel processing of the separate populations, but it also produced better results than the ordinal canonical genetic algorithm. The isolation of sub-populations has lead to significant research into Island Genetic

Algorithms (iGAs), whose term denotes a population split into many semi-isolated islands. There are several different but equally accepted hypotheses about why iGAs are able to evolve better then Canonical GAs (CGA). Generally the subdivision of the population forms some basis of speciation, whereby separated groups of individuals may evolve genetically similar individuals likened to species in nature. The mixing of properties from each species during the migration phase leads to an injection of new schema into the evolution process, allowing iGAs to maintain diversity better than CGAs.

The implementation of an iGA, requires the specification of several additional parameters including the number of sub-populations or demes, the number of generations that occur between migration periods, termed the migration interval, and the number of members allowed to migrate from one population to another, termed migration size. Belding [86] and Cantú-Paz [87] have analysed the importance of deme size, migration interval and rate on the convergence and solution quality of iGAs, finding that iGAs are able to sustain several solutions if the migration process is moderately applied. A description of the iGA implementation into the existing GA is shown in Figure 21

```
1. Initialise template DNA.
2. Initialise $NDeme$ Genetic Algorithms with template DNA.
3. Initialise $ImmigrationControl$ for $NDeme$ islands.
4. For $i = 0$ to $i = NGen - 1$
    (a) For $j = 0$ to $j = NDeme - 1$
        i. Receive $Immigrants$ from $ImmigrationControl$.
        ii. Replace $Size(Immigrants)$ with $Immigrants$ at random in $Parents$.
        iii. Evolve $GA[j]$ One Generation.
        iv. Save Global Stats.
        v. If reached $migrationInterval$
            A. Select $NMig$ best members and copy to $ImmigrationControl$.
5. Save Global Stats.
6. End.
```

Figure 21: Implementation of the Island Genetic Algorithm

An iGA scheme was tested on the bump problem with linear rank scaling for breeding selection within the subpopulation. Migration interval, migration rates and deme size were investigated for the $n = 20$ bump problem. Two ring based topologies

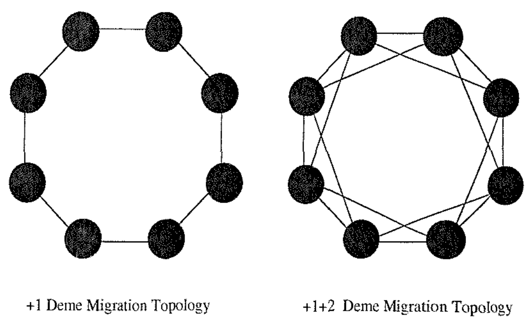

Figure 22: Deme Topology and Migration Strategies

were used for migration coordination, with demes sharing members with other demes immediately topologically adjacent for the first test case, and with demes within a topological neighbourhood of two as illustrated in figure 22. At the heart of the migration process, is the immigration control algorithm that is responsible for coordinating Genome migration as dictated through the topology description. The immigration control process implemented is shown in Figure 23.

The results of the iGA implementation on the $n = 20$ bump problem is shown in Table 7 based on a total population size of 300 members. Superior results to that of the basic GA are observed, particularly at migration intervals between 10 and 20 generations. For larger deme sizes for a chosen migration rate of 0.2, at migration intervals of 10 generations, a small improvement in optimal fitness can be obtained but at a significantly larger evaluation cost. This would imply that although more populations are used, a better and more stable optimal solution can be obtained in comparison to a standard GA implementation. For high dimensional problems, the associated cost vs fitness obtained by employing several additional demes, will converge to that of a standard single population GA implementation.

```
1. Initialise for $NDeme$ islands.
2. Create $islands$=LinkedList[$NDeme$].
3. Create $ImmigrationStrategy$={$1, -1, 2, -2$}.
4. On $AddImmigrants$ Function Call(int $SendingGA$, LinkedList $immigrants$)
    (a) For $i = 0$ to $i < Size(ImmigrationStrategy)$
        i. while $Size(immigrants) > 0$
            A. $targetGA$=$SendingGA + ImmigrationStrategy[i]$.
            B. If $targetGA >= NDeme$ then $targetGA- = NDeme$.
            C. If $targetGA < 0$ then $targetGA+ = NDeme$.
            D. Remove $First(immigrants)$ and add to $islands[targetGA]$.
    (b) Return.
5. On $GetImmigrants$ Function Call(int $ReceivingGA$)
    (a) $Result = islands[ReceivingGA]$.
    (b) Clear $islands[ReceivingGA]$.
    (c) Return($Result$).
6. End.
```

Figure 23: Immigration Control Algorithm for iGA (+1+2 Strategy)

Migration	Migration Rate			
Interval	0.1	0.2	0.4	0.5
5	0.46	0.50	0.46	0.35
10	0.55	0.63	0.51	0.43
20	0.51	0.62	0.38	0.42
50	0.38	0.37	0.42	0.39

Table 7: Effect of Migration Rate and Interval on an iGA (6 demes of size 50)

4.3.2 The Concept of Niche Formation

Speciation in natural ecosystems is the process whereby a single species differentiates into several different species to occupy different niches. Such a process aids the evolution cycle to become more adaptive to changes in climates/environments through the preservation of diversity. In GAs, diversification can be aided by restricting mating between different species (defined by niche occupation or through a separate DNA specification) where niches represent different peaks of local optima. Diversification is particularly important where more than one maxima peak is required. Unfortunately a GA's population will naturally converge on a single peak due to genetic drift [88]. Three approaches that are generally used to influence population diversity are species control, population crowding prevention, and to share the payoff associated with a

Number	Deme	Size
Demes	20	75
5	0.26	0.64
10	0.32	0.64
20	0.35	0.69

Table 8: Importance of Deme Size on an iGA ($mig_{int} = 20$, $mig_{rate} = 0.2$)

niche.

Species Control

Species control restricts the way in which a member mates with another. Restrictive mating is usually employed to contain a niche, either by preventing members of a dominated peak mating with members outside the peak, or if several species with different DNA makeups are present, to prevent some key species characteristic genes being exchanged with different species. The second case can be used if several different designs are considered that share some gene similarity, and diversity is required so that initially weaker design types are not eliminated from the design too early. The implementation of non-interbreeding species is easily implemented using a similarity template called an external scheme [88]. The concept of using restrictive mating to contain a niche, requires identification of the niche and niche size.

Population Crowding Control

Population crowding was first proposed by DeJong and implemented by Goldberg [11] and Baker [79]. In the original crowding model, each offspring is compared to a number of parent members replacing the most similar parent. The number of parent members used in each comparison is termed the crowd-size, and can range from 1 to N where N is the population size. Although this method shares many basic foundations with nature, it was unable to maintain more than two peaks on most multi modal problems. This was demonstrated by Mahfoud [89], who showed that the scheme exhibited a large degree of replacement error (child replacing a parent that had better fitness), which degraded GA performance through increased schema noise. Mahfoud proposed a new crowding scheme called 'deterministic crowding', where offspring replaced their nearest parent if they performed better. To reduce parental replacement error, a similarity check is achieved by calculating the Euclidean distance between members, as defined by the absolute difference between chromosome normalised values. This scheme therefore imposed its own deterministic selection scheme offering significant parallelization opportunities.

Sharing Niche Payoff

Members occupying the same niche can be made to share the fitness payoff between them until the niche reaches its carrying capacity. At this point, the fitness payoff of occupying that niche is less attractive than that of other niches.

The shared fitness of individual i is given by

$$SharedFitness = \frac{ActualFitness}{\sum_j s(d_{ij})} \tag{80}$$

where a basic sharing function $s(d_{ij})$ [90] may be defined as

$$s(d_{ij}) = \{ \begin{array}{cc} 1 - (d_{ij}/\sigma_{share}) & if d_{ij} < \sigma_{share} \\ 0 & otherwise \end{array} \tag{81}$$

and the distance d_{ij} is usually the separating distance between two individuals in Euclidian space.

This method can maintain diversity better than crowding [88][91], but is more difficult to implement as the niche locations and radii are usually unknown. Members closely grouped together have their fitness de-rated by how close they are to the fitness centre and the number of individuals within a niche radius. If the niche is significantly good, a group of members will remain on that niche allowing other members to continue the search. Speciation techniques can also be employed to prevent members of a niche mating with members outside, preventing population dilution and encouraging increased search within the niche.

Several methods for implementing niche sharing have been proposed, with earlier approaches based on the assumption that niches are known and distributed evenly throughout the solution space. Implementation of niche schemes are particularly difficult where many local maxima are located near the global optimum. The location of the optima will be dependent on members of the population being left once all the local niches are filled [89]. A sequential niche method has been proposed [92] involving many sequential runs of the GA, each locating one niche. The fitness function is then modified on successive runs to cancel out the previous niche.

The original sharing scheme proposed by Goldberg [88] requires knowledge of the number of niches 'a priori'. Such prior knowledge is obviously not available for the majority of problems found in engineering. Yin and Germay [93] proposed the use of an adaptive cluster identification algorithm given in Appendix A, based around the KMEANS clustering algorithm. Here a set of k members are allocated a cluster each. The pairwise distances between all clusters are computed, and if they are less than a predetermined distance, the cluster is collapsed to another. Merging is continued until all centroid's are separated by at least a predetermined value. The rest of the population are then assigned to their nearest niche, and cluster merging or separation is continued until the cluster constraints are met.

Application of Niching to Control Population Diversity

A larger population size was required for deterministic crowding to sufficiently fill niches whilst maintaining diversity. Using only the deterministic selection pressure, convergence was found to be too slow for practical use. Addition of further selection pressure was attempted, first using tournament selection which resulted in premature convergence. With application of rank selection with deterministic crowding, the resultant population found is shown in Figure 24 with several niches identified, including the optimum.

Sharing and tournament selection are not ideally suited together [82] (Tournament selection wants to select the best members, while sharing wants to de-rate members). The adaptive KMEANS algorithm was implemented using rank selection. A population size of 100 members was used to ensure that the niches were adequately covered. The convergence of the solution is shown in figure 25, showing the optimum was found

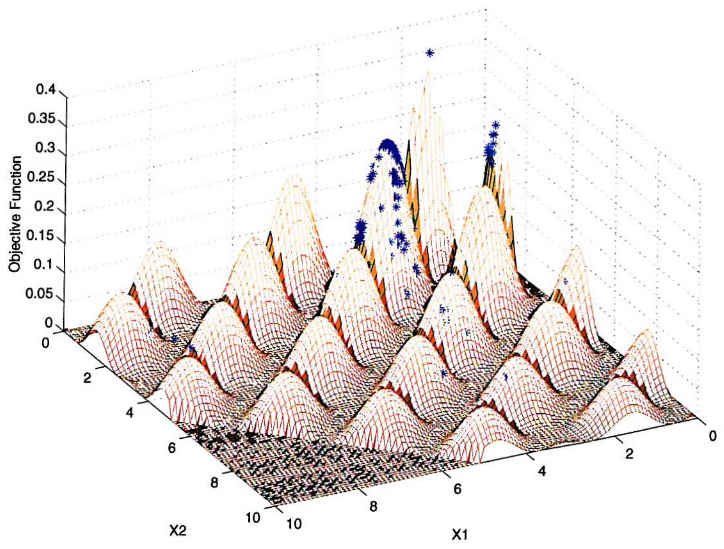

Figure 24: Optimal Found with Deterministic Crowding (popSize 100, nGen 300)

on all five runs, the second series of lines show the average fitness of the population. The final population is shown in Figure 26.

A trace of population best members throughout the GA evolution is shown in Figure 27, illustrating how sharing allows the GA to move from one peak to the next.

4.3.3 Specification of Key GA Parameters

In the simple GA model, key parameters such as steady state population size, crossover rate, mutation rate, and fitness scaling, needed to be carefully implemented. With the inclusion of niching, large population sizes are required such that the niches found can be adequately populated without significant loss of diversity in the remaining population. For the two dimensional bump problem, population sizes of up to 100 were required to significantly improve the probability of locating the optimum. This is an increase by a factor of 2-3 on original hit and miss attempts. Since the number of design variables will also dictate the number of niches available within the search space, population size still depends partially on the search dimension size. Some guidance to the size of populations, crossover and mutation rates to be considered are given by DeJong [94] and Back [80], although pre-testing final parameter values

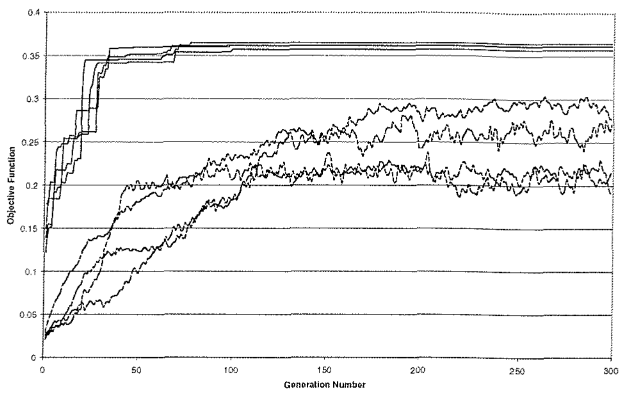

Figure 25: Convergence of the Bump Problem with Niching (popSize 100)

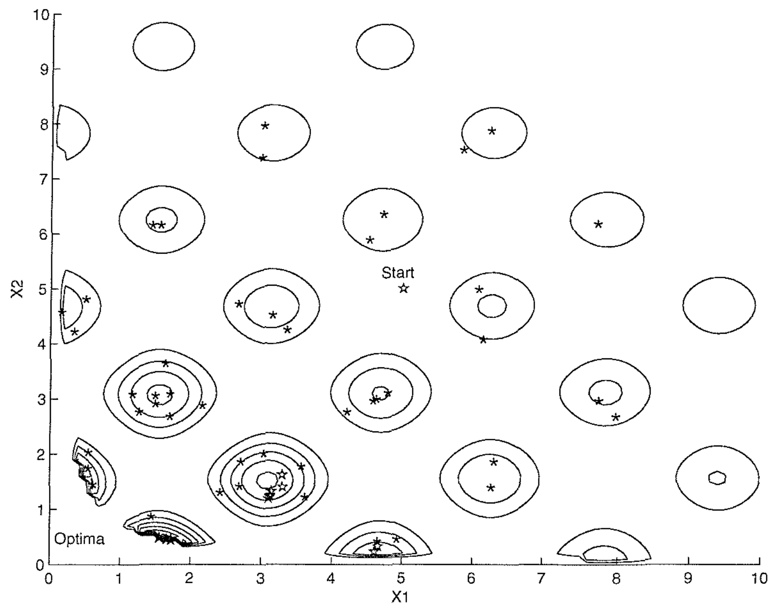

Figure 26: Effect of Niching (popSize 100, nGen 300)

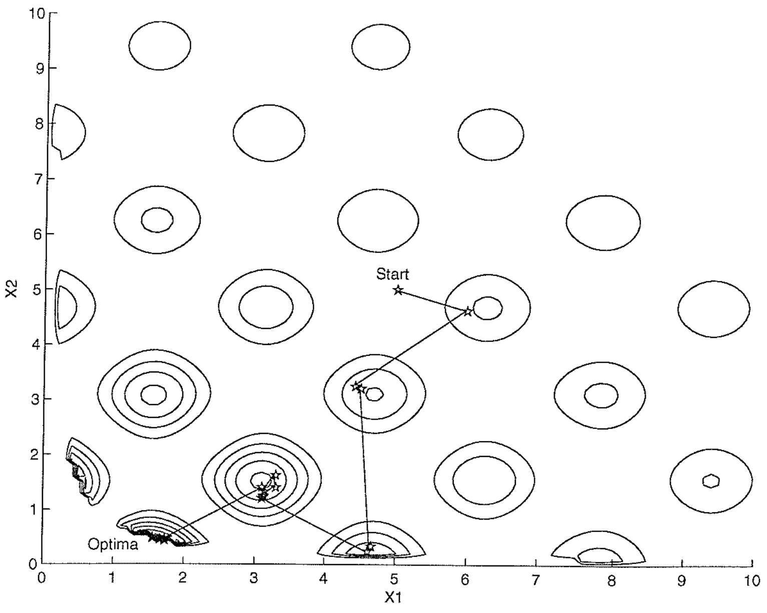

Figure 27: Search Trace of Best Members (popSize 100, nGen 300)

on problems such as the bump case with the same number of dimensions is always advisable.

The adoption of deterministic crowding reduces the chance of good parents being replaced by poor offspring. In the original deterministic crowding scheme without further selection pressure, the maximum crossover rate of 1.0 was used. When some additional selection pressure was added, this rate was reduced to help niche formation. In general, crossover rates between 0.8 to 1.0 are most constructive for the deterministic crowding schemes. Mutation rates that ranged between 0.5% and 2% were found to be most constructive towards the evolution process.

4.3.4 Analysis of Speciation and Optima Finding

Diversification of population DNA has been a critical characteristic of Genetic Algorithms in finding the optima in multi-modal landscapes. Speciation methods such as crowding and fitness sharing, can help to maintain diversity while the generation matures preventing population takeover. However in their implementation, it has been clear that the adoption of such techniques still cannot guarantee success in finding the optima. Due to the nature of the binary crossover operator used, a new species generally is created through cross-breeding two other species. The more different the parent species, the more combinations of new species there are that can be developed, and the further away from their parents they could become.

To gain some insight into why the optima can remain elusive despite the diversity of the population, an analysis of how new species are formed and survive in the presence of fitness sharing has been performed.

In this analysis, a new species is said to have been created if

$$\sigma_{Child_{i,j}} > D_{species} \tag{82}$$

where $\sigma_{Child_{i,j}}$ is the Euclidian distance of child from its parents i, and j, and $D_{species}$ is the minimum distance between two species.

For the 2D bump problem, an analysis of new species creation has been made using a genetic algorithm with a population size of 50, crossover rate of 0.85, linear rank fitness assignment and SUS selection. Mutation and inversion was not used, and a single elitist member survival strategy has been implemented. Figure 28, shows the number of species present in each generation, divided into the number that survived to the next generation, and the number that failed to be selected to become parents.

The rate of population convergence is clear in this figure, with population takeover occurring after just 10 generations due to the high crossover rate used. A high rate at which new species are formed at the beginning of the evolution process is shown, as well as the large probability of these new species surviving.

The same analysis is presented in Figure 29, with a fitness sharing scheme applied to the selection process. Once again a large number of new species has quickly been established early on in the evolution process, but the effect of species diversification through the application of fitness sharing has maintained this diversity throughout the maturing population. Despite the increased diversity, an equilibrium was soon established between creation of new species, and survival to the next generation. Through investigation into the family tree of an optimum, on four out of five occasions, the optimal niche was found through cross-breeding two similar species within the first

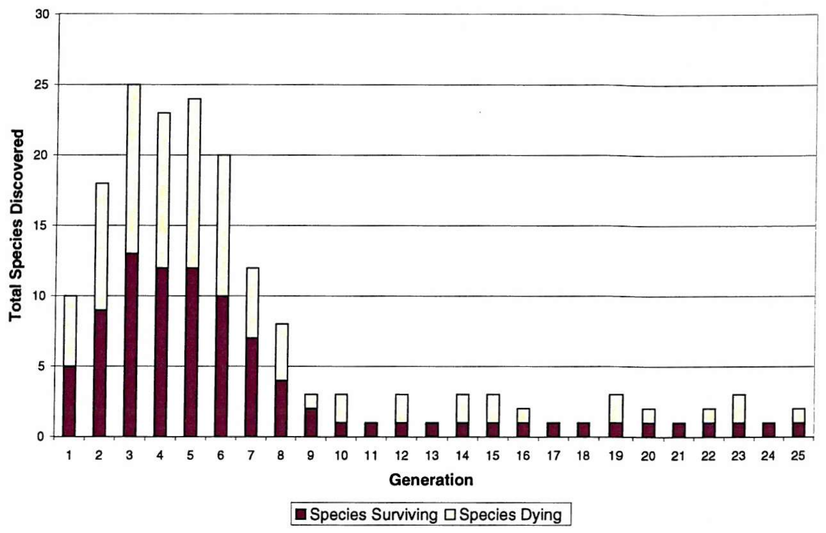

Figure 28: Rate of Species Creation for a Non-Niching GA

six generations where the chance of a new species surviving remained high. In the fifth instance, the optimal niche was established at generation 37, although this niche had been hit with non-surviving children on 27 instances preceding this. This final instance arose more out of an evolution of fluke, rather than a result of continuous gradual adaption.

Analysis of the fitness of new species created in the presence of fitness sharing throughout the evolution process shown in Figure 30, illustrates that despite the fitness of its parents, new species are likely to rank less favourably in the selection process. With a low population maturity, species created were found to share a fitness comparable to the population average, and therefore likely to rank amongst selected parents. However, the population average fitness soon increases as the population matures, and by generation five it becomes difficult for new species to survive parent selection.

The problem of species creation is acute for the bump problem, since successful and non-successful candidates can lie very closely together. For a GA that is only able to evolve members within a single environment, this problem can severely affect its performance in finding the optimum. The rate of species death could be reduced by encouraging species to inter-breed. However this will not aid the establishment of

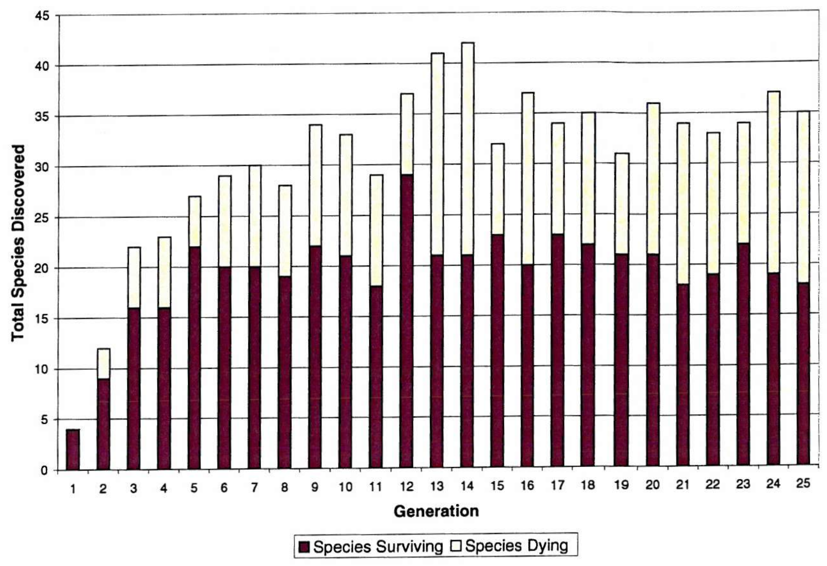

Figure 29: Rate of Species Creation for a Niching GA

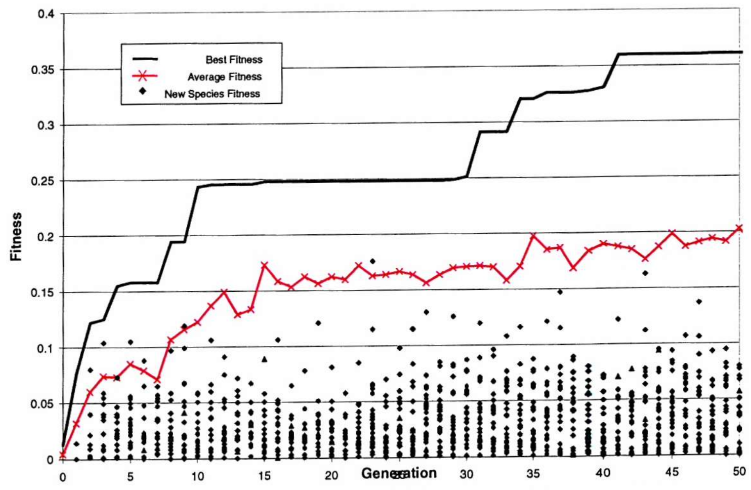

Figure 30: Fitness Distribution of New Species vs Overall Best and Average Fitness

new species which has been the aim of the work presented here. To encourage the GA to allow tolerance towards new species during parental selection, environmental changes will need to be explored, to provide some sort of temporary stepping stone between the creation and long term establishment of species.

4.4 Summary

A Genetic Algorithm has been established, based on the Canonical GA by combining both island migration schemes with fitness sharing. This work has presented evidence indicating that resource sharing allows newly formed species to survive and contribute better to the evolutionary process, because new members are able to compare better with the average population fitness. The introduction of resource sharing has also prevented the problem of population takeover, producing a significantly more robust algorithm on which to base research into adaptivity. A downside consequence of this increased robustness, is the significant increase in evaluation cost associated with the larger population sizes required for both island models and resource sharing. Near optimal solutions to the 20D bump problem can be found to a large degree of robustness with a population size of 300 members across four demes, each requiring at least 50 generations to converge. This leads to a cost in excess of 1000 times that indicated by Jameson *et al.* [25] in his Adjoint formulation approach to airfoil design. Although higher quality solutions may be found using Genetic Algorithms, significant computing resources are required for use in engineering design such as shape optimisation.

Chapter 5

Distributing Expensive Objective Function Evaluations over an Office PC Network

The use of Genetic Algorithms in aerodynamic optimisation necessitates the need for high performance computational facilities. Genetic Algorithms are implicitly parallel, and this chapter discusses the algorithms used to distribute some of the computational aspects of GAs across an office PC network, its distributed performance, and the reliability and scalability of the PC network for aerodynamic computation.

5.1 Introduction to Parallel Genetic Algorithms

Parallel genetic algorithms (PGAs) have been used to solve computationally expensive problems [34]. Such problems need a bigger population, and hence require more processing resources. The motivation behind early studies into PGAs was to reduce the processing time needed to reach an acceptable solution. This was achieved by implementing GAs on parallel architectures. In some cases, it was noted that PGAs found better solutions than comparably sized serial GAs.

Shape optimisation using CFD to evaluate the objective function, provides several motivations for using PGAs. Parallel architectures are widely incorporated in the field of CFD [95][70] with many resources solely configured for the purpose of CFD analysis and development. CFD parallelisation is necessary, not only to reduce the computing time required to reach a solution, but also to provide the computer memory resources needed to store the large computational domain and millions of flow variables associated with solving many complex problems, while maintaining some degree of effective cache utilisation. parallelisation of the GA can become necessary when the evaluation of the population requires large computational resources. Thus, the problem of interfacing a GA with CFD can draw on three types of parallel resources:

- a Serial GA evaluating population members in-turn using a parallelised evaluation function,
- a PGA evaluating single or clusters of members over several processor nodes using a serial evaluation function,
- a hybrid interface using a PGA, using further processor nodes for parallel implementation of the evaluation function.

In this chapter, the global parallelisation of Genetic Algorithms involving computationally expensive fitness functions ,will be investigated over a medium size network of existing Personal Computers (PCs).

5.2 Utilising Existing Computational Resources to Create a Commodity PC Network Suitable for Fast CFD Computation

Over the past decade, the cost of computing has reduced significantly and the convergence of the price and performance of high-end workstations with affordable PCs has been particularly rapid over the past few years. Through such advances in computing technology it is now feasible to consider the long-term role of PC's in the field of CFD.

Through the utilisation of network technology it is possible to connect clusters of PCs together. When used with parallel and distributed computing software libraries such as MPI [96], it is possible to obtain Super-Computer level machines at the fraction of the cost [97] of their stand-alone predecessor machines such as the CRAY T3E. These machines are burdened with enormously high setup and support costs, whereas a PC cluster offers a minimal cost environment with the potential for incremental improvements in performance.

This section explores the development and performance evaluation of such a system, developed from an existing cluster of teaching PCs used for the Ship Science courses at the University of Southampton. The teaching cluster was purchased to service the need of students for dedicated ship design software, CAD packages, programming skills and general office software. For a small cost (£9,000), the network was adapted to allow dual use - providing a teaching resource during term time between 8 am and 9 pm and for the remainder acting as a networked cluster available for carrying out CFD computations.

5.2.1 Creation of a Dual use computational facility

The original network of PCs consisted of 52 PC units of at least 350Mhz processing speed, and 64Mb+ RAM available, connected via 10MHz Ethernet directly to the NT4 server through 10MHz switches. This network is then connected to the main UNIX computing facility via an Internet connection. Red Hat Linux 6 was selected as the target operating system for the high performance network, for its increased stability over Windows, its ease of porting and similar environment to UNIX. A dual boot facility was targeted to accommodate both teaching support through NT and high performance processing using Linux.

Two high-end dual processing PC workstations were purchased to provide a base for building computationally intensive problems, which could be extended later in future purchasing plans. One of these workstations was chosen to serve as the Linux server, as well as a desktop workstation during the daytime. To accommodate a finite budget, a hardware trade-off was made in favour of network requirements for high processor speed, 10/100 Ethernet and a fast 36Gb Ultra2 Wide SCSI hard drive. An

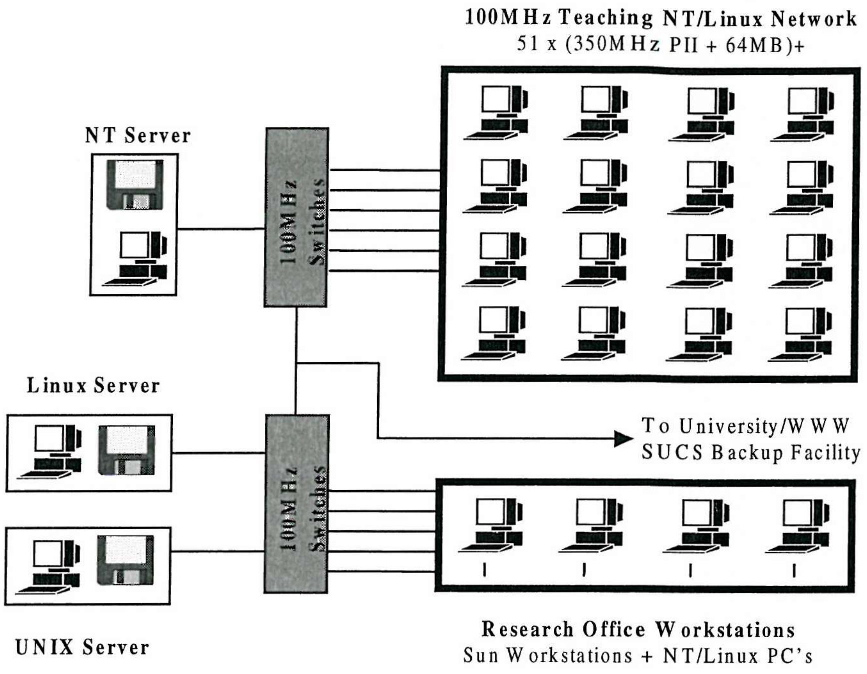
100MHz Teaching NT/Linux Network
51 x (350MHz PII + 64MB)+
NT Server
100MHz Switches
Linux Server
100MHz Switches
To University/WWW
SUCS Backup Facility
UNIX Server
Research Office Workstations
Sun Workstations + NT/Linux PC's

Figure 31: Schematic Diagram of the Departmental Teaching and Research Network

affordable 512Mb of memory was added to both workstations such that small CFD problems could utilise the dual 500MHz PIII processors. A file server was created using NFS, and user and password authentication through the existing UNIX server. File backup is currently carried out through the UNIX file server, but will later use the University global backup facility via the Intranet, significantly reducing costs. The speedup scalability of the commodity cluster is mainly limited to network speed performance [98], originally 10MHz, and an upgrade to an affordable 100MHz was carried out as shown in Figure 31, so that high performance CFD computations could be realised. The original computing facility was based around two networks, the NT and UNIX, which remained as separate entities so that the performance of one would not degrade the other. Communication between networks was carried out through the external communication line that connected the UNIX network to the University Intranet and outside world. High speed cabling that could accommodate a faster 100MHz network already existed for the PC cluster, and its full network upgrade to

100MHz required only new network switches. To allow scalability of the commodity cluster beyond the 52 PCs housed in a single computer room, any new PCs entering the department for research use, should also be accommodated. The research offices are networked through the UNIX 100MHz switches, and a simple upgrade of network cabling was needed to offer full scalability.

The Linux server carries out parallel process scheduling to avoid either processor or network overload. The MPI libraries are in current use for three in-house flow solvers, as well as for a parallel processor Genetic Algorithm and would form the main software for use on the network.

5.2.2 Evaluation of the computational resource

In order to evaluate the performance of the new computing resource, tests have been conducted using two different computational problems typical of the day-to-day computational work carried out within the department. An unstructured Euler solver [99], has been ported into Linux using GNU gcc compiler. The MPICH 1.2.2 library from Argonne National Laboratory was used for message passing between mesh partitions distributed on separate processors. A distributed Genetic Algorithm has been implemented, and tested for optimisation using a modified bump problem that necessitates the need for efficient inter-processor communication. MPICH is used to facilitate a global Master-Slave distributed implementation of the GA. Porting into Linux was achieved using GNU g++, recompiling source code for both the Euler solver and GA. Only minor porting problems were found with the Euler solver where the communication buffers created using *MPI_Struct* procedures, required some extra attention to ensure that the data was correctly aligned within the buffer, without overwriting itself.

The Message Passing Model

The three main types of parallel architectures are: vector machines, shared memory and distributed memory machines. Vector machines exploit vector manipulations such as vector multiplication and matrix inversion. Shared memory machines, group processing units around a global memory bank. The control of memory access is processor based rather than user based, simplifying the process of parallelising programs which share a common data base. The complexity of memory access increases as the number of shared processors increases, which has tended to limit their scalability. Distributed machines group processor units that use their own local memory. Their scalability is simplified as there are no conflicts with memory access, the sharing of memory between processors is defined by the software, and so the development of distributed programs is more difficult than shared memory programs.

The implementation of the control strategy is based on the use of applications developed within Ship Science using MPI and C/C++. Distributed memory architectures are more scalable than shared memory architectures due to the memory being local to a processing unit. They may also be implemented on shared memory architectures whereas the opposite is not true.

The message passing model is a distributed memory model which differs from the *Single Instruction Multiple Data* approach, in that each process is associated with an individual data set on which non-identical instructions may be performed. Processes may communicate with any other through executions of commands on both the 'sending' and 'receiving' processes. The advantages of the model have been summarised [100] as allowing compatibility with many different architectures; a great deal of control compared with data parallelism and compiler based methods; and an increase in processor performance that can be realised on cache based machines[101]. This is the most common choice of message passing implementation, and is used in this work.

The *Message Passing Interface*[102] (MPI) is the result of collaboration by many developers of message passing models, which aimed at standardising previous implementations. The result is a library of functions which, at their lowest level provide the basic **send** and **receive** functions necessary for the implementation of the message passing model, whilst at a higher level, provide routines which aid the developer in the efficient parallelism, debugging, and monitoring of codes. Each process executes identical codes within which logical statements are used to separate process tasks. This approach has been termed *Single Program Multiple Data* which is a subset of the *Multiple Instruction Multiple Data* model.

Distributed Implementation of the Unstructured Euler Solver

For the unstructured Euler solver introduced in Chapter 3.6, the parallel strategy for the explicit field method is straightforward, in that the domain is partitioned according to the number of processors available. A satisfactory decomposition is achieved through the Jostle [103] graph-partitioning program. Using a knowledge of the Cell ordering, each cell is allocated to a process, followed by the remaining geometrical objects required to describe the Cells.

On partition interfaces, the Cells are duplicated in order to complete their descriptions on both partitions, and to provide enough information for the numerical flux calculation. To ensure that flux calculations are not duplicated on neighbouring processes, a root processor is assigned to each interface Cell to carry out its flux calculations. The locations of interface cells that shadow the root Cell on neighbouring partitions are assigned a 'shadow' flag, holding the location of the root Cell.

To implement the parallel solver algorithm given in Figure 32, the partitioned grid is based around a Node to Node connectivity map, joined together by Edges. Control volumes are constructed around each of the Nodes, forming the Cells with a dual grid associating each Cell face cutting between a Node-Node Edge on the original grid. The calculation of the face fluxes, is carried out by an Edge based loop, updating residuals on all nodes except those that are shadows. Contributing boundary faces to the node residuals are then included.

A semi-scheduled message-passing algorithm is implemented to update the shadow residuals residing on neighbouring partitions. Essentially the sending of residuals from the host partition is implemented through a message-scheduling algorithm with non-blocking communication. This allows the send process to start before a matching receive is posted, minimising dead-time events where processes are held waiting for recipient partitions. An illustrative algorithm for the semi-scheduled message passing algorithm is shown in figure 33

A node sweep is used to update all non-shadow nodes, with root partitioned nodes communicated to their adjacent shadows via the same semi-scheduled algorithm as before. Residual statistics are updated on the master process, which determines and

```
1. Input & Distribute Data
2. For $i = 1$ to $i =$ ROOT EDGES :
    (a) Evaluate control volume face normal $\hat{n}_i$ and area $S_i$.
    (b) Evaluate $\delta V_\Omega$ associated with control volume face and distribute to edge nodes.
3. End Loop.
4. For $i = 1$ to $i = (no.nodes)$ :
    (a) Initialise Residual $\delta U_i = 0.0$
    (b) Store Runge-Kutta Reference Values $U_i^0 = U_i$
5. End Loop.
6. Initialise FLAG = FALSE
7. Initialise COUNT = 0
8. Do :
    (a) COUNT = COUNT + 1
    (b) For $l = 1$ to $l = 4$ :
        i. For $k = 1$ to $k =$ ROOT EDGES :
            A. Evaluate Numerical Flux $f^*$ using Roe's Approximate Riemann Solver in the normal direction $\hat{n}_k$.
            B. Update edge node residuals. $\delta U_i = \delta U_i + S_k f^*$   $\delta U_j = \delta U_j - S_k f^*$
        ii. End Loop.
        iii. For $i = 1$ to $i =$ ROOT FACES :
            • If face lies on solid boundary :
                A. Evaluate flux $\vec{F}$ with zero mass flow imposed.
                B. Evaluate directed face areas $\vec{S}_i$.
                C. Distribute contribution $\vec{S}_i \cdot \vec{F}$ to face nodes.
            • If face lies on open boundary :
                A. Evaluate face normal $\hat{n}_i$ and areas $S_i$.
                B. Evaluate numerical flux $f^*$ using free-stream conditions as the appropriate Riemann state in the direction of the face normal $\hat{n}_i$.
                C. Distribute contribution $S_i f^*$ to face nodes.
        iv. End Loop.
        v. Pass Residual Contributions From Duplicate Nodes to Root Nodes
        vi. For $i = 1$ to $i =$ ROOT NODES :
            • Evaluate $\Delta t$.
            • Update Conserved Variables $U_i^l = U_i^0 - \frac{\alpha_l \Delta t}{V_\Omega} \delta U_i$
            • Reset Residual $\delta U_i = 0.0$
        vii. End Loop.
        viii. Pass Updated Conserved Variables From Root to Duplicate Nodes
    (c) End Loop.
    (d) Reset Maximum Error $\delta U_{max} = 0.0$
    (e) For $i = 1$ to $i =$ ROOT NODES :
        i. If $|U_i - U_i^0| \geq \delta U_{max}$ : $\delta U_{max} = |U_i - U_i^0|$
        ii. Update Runge-Kutta Reference Variables $U_i^0 = U_i$
    (f) End Loop.
    (g) Communicate Local Maximum Errors to MASTER Process
    (h) Check for Convergence on MASTER Process:
        i. If $\delta U_{max} \leq \delta U_{convergence}$ FLAG=TRUE
        ii. Else
            A. If COUNT = DUMP
                • FLAG Gather and Output Data
                • Reset output counter COUNT = 0
    (i) Distribute Convergence FLAG
9. While FLAG = FALSE
10. Gather and Output Data
```

Figure 32: Distributed Upwind Algorithm

```
1. Initialise Integer Send Flag $SendFlag_{(int)} = NULL$.
2. Initialise Message Send Flag $SendFlag_{(mssg)} = NULL$.
3. Fill Message Lists.
4. For $i = 1$ to $i = N - 1$
    (a) Post an integer **Receive** (size of message $q$) from any Process $q$ where $0 \leq q < (N - 1)$ and $q \neq p$, with $FLAG = RecvFlag_{(int)}$.
    (b) **Wait** for $SendFlag_{(int)}$ to return NULL OR TRUE.
    (c) **Wait** for $SendFlag_{(mssg)}$ to return NULL OR TRUE.
    (d) Post an integer **Send** (size of message $i$) to process $i$ above $p$ with $FLAG = SendFlag_{(int)}$.
    (e) **Wait** for $RecvFlag_{(int)}$ to return TRUE.
    (f) If size of message $q > 0$
        i. Allocate Buffer.
        ii. Post a message **Receive** from the Process $q$ with $FLAG = RecvFlag_{(mssg)}$.
    (g) If size of message $i > 0$
        i. Post a message **Send** to Process $i$ above $p$ with $FLAG = SendFlag_{(mssg)}$.
    (h) If size of message $i = 0$
        i. Set $SendFlag_{(mssg)} = NULL$.
    (i) If size of message $q > 0$
        i. **Wait** for $RecvFlag_{(mssg)}$ to return TRUE.
        ii. Empty Buffer.
5. End Loop.
6. **Wait** for $SendFlag_{(int)}$ to return NULL OR TRUE.
7. **Wait** for $SendFlag_{(mssg)}$ to return NULL OR TRUE.
```

Figure 33: Semi-Scheduled Message Passing Algorithm

implements convergence, and full field-data checkpointing requirements. Checkpointing is achieved through a full sweep of processors sending all flow variables to the master for storage.

Distribution performance on the Linux cluster

To determine the efficiency of the network for Euler solver computation, network performance timings were measured during the computation of a 2D NACA0012 wing illustrated in Figure 34. The two dimensional flow effect is simulated by enclosing the wing ends by boundary walls shown in Figure 35 which shows the boundaries of the computational domain. The discretised computational domain shown in Figure 36, consisted of a three-dimensional grid made out of 48672 control volume Cells. For a flow simulation at $M = 0.85$, and a wing angle of attach of 0^o, the pressure contours given in Figure 37 are obtained. For the performance test, timing measurements were

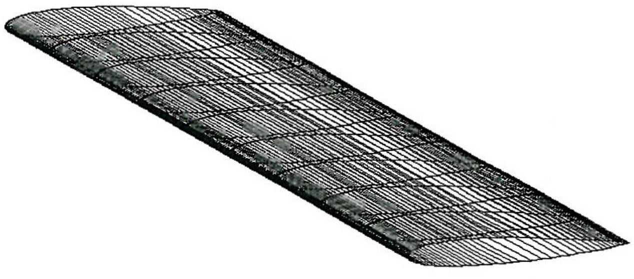

Figure 34: NACA 0012 Uniform Wing

based on 1000 flow solver iterations of the time integration loop, for each partition topology. Performance tests were made on the original 10MHz network before upgrade as well as on the new 100MHz network. Separate tests were conducted, allowing for regular checkpointing at 20 iteration intervals on both networks.

Speedup performance results are shown in Figures 38 and 39, where speedup is defined as

$$Speedup = \frac{Net\ time\ to\ compute\ flow\ solution\ with\ N\ procs}{Net\ solution\ time\ for\ single\ processor}, \tag{83}$$

and iteration efficiency as

$$\eta_{itteration} = \frac{Net\ time\ taken\ within\ itteration\ loop}{Net\ solution\ time}, \tag{84}$$

and total efficiency:

$$\eta_{itteration} = \frac{Net\ calculation\ time\ excluding\ MPI\ operations}{Net\ solution\ time}. \tag{85}$$

The single processor result used in Equations 83,84, and 85, is based on the same distributed algorithm given in Figure 33 as for the multi processor results. The speedup measured shows substantial gain in computation speed for up to 16 processors at

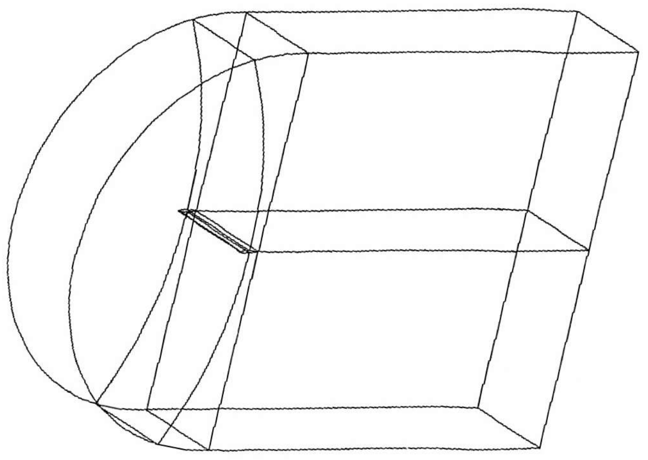

Figure 35: Computational Domain for NACA 0012 Uniform Wing

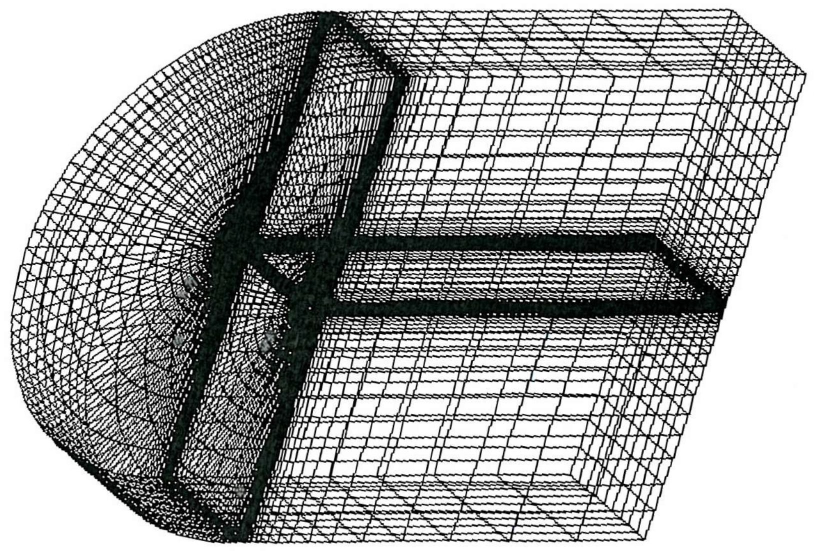

Figure 36: NACA 0012 Uniform Wing Mesh

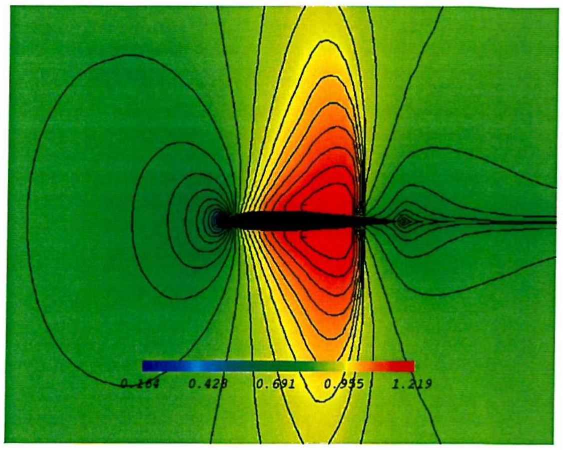

Figure 37: Mach Contours NACA 0012 $M_\infty = 0.85$

100MHz-network speed. Even with regular saving of flow values to disk, a reasonable performance is sustained. The results from the original 10MHz network highlight the need for good network speed for such solutions. Closer evaluation of iteration efficiency of the cluster, reveals that further improvements to speedup performance can be sought if the size of the partition domains were maximised for the performance tests. By partitioning the domain to maximise processor memory use, iteration efficiency can be further recovered. The maximum partition size for an unstructured grid that can adequately be used on each processor, consists of approximately 30,000 nodes. Using just 16 of the 52 available processors a 0.5M cell solution can be expected with good speedup. The maximum domain size available through the entire cluster is approximately 1.5-2M cells.

Distribution Performance for a Genetic Algorithm

In the implementation considered here, the 20D Bump problem used in Section 4.1.1, has been modified such that the time taken to compute the objective function is $O(T_{bump})$, where T_{bump} is randomly perturbed from a defined amount. In this exercise, T_{bump} is defined as 1 minute with a random perturbation of up to 20%. This represents a typical time for evaluating a single airfoil using XFoil. This additional feature places the emphasis of distributed GA performance on asynchronous communication

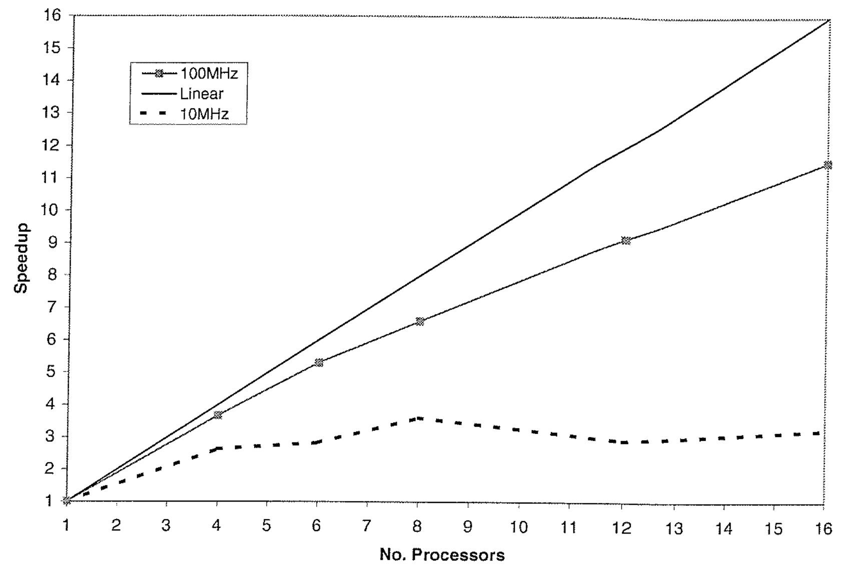

Figure 38: Speedup performance of Euler Solver on PC Cluster

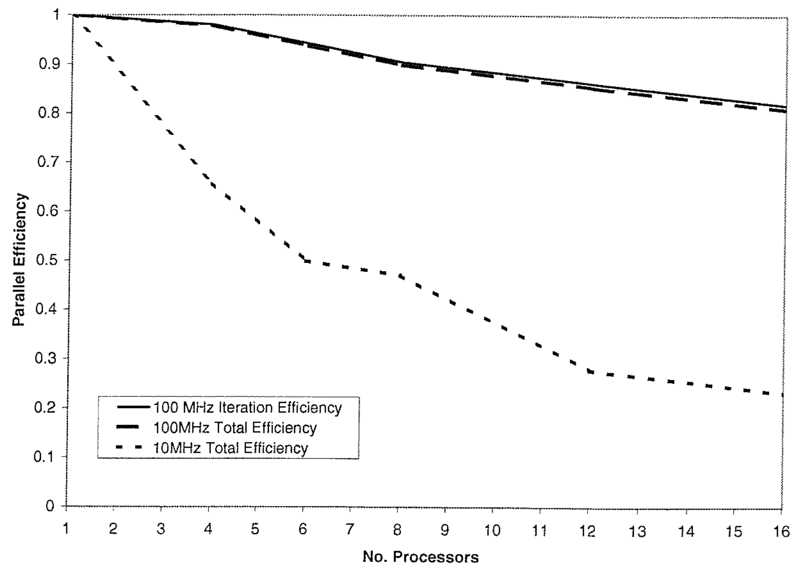

Figure 39: Parallel efficiency of Euler Solver on PC Cluster

implementation.

A global parallelisation strategy was originally used to distribute the evaluation phase of the population members only, based on the Master-Slave implementation given in [11]. This scheme was constructed using the MPICH library distributing an initial subset of the population to the available processors, and then passing the remainder on to each processor in turn on its completion with its current member. Due to the non-synchronous time dependent nature of the modified bump function, the efficiency of this method is degraded on larger cluster sizes when all members have been passed for evaluation, and processors are left waiting for remainder slaves to complete before the next generation can be created and the distribution process continued. For a population size of 200, distributed using 50 slave processors, distribution inefficiencies of up to 35% were found on some generations.

To alleviate this problem, an asynchronous strategy was sought that decouples the evaluation processes from the synchronised selection phase of the GA. Asynchronisation was achieved based on the island Genetic Algorithm (iGA) used by Doorly [34]. In the iGA implementation, sub-populations known as demes, are used to partition the population, each evolving independently with occasional exchange of genetic information between demes via a process termed migration. In this implementation, all evolving demes are contained within the master process, with members sent to slaves for evaluation using an asynchronous communication strategy. When a deme has passed all its members through for evaluation, but remains idle waiting for the last few evaluated members to return from their slave processors, members from the next deme are used to occupy idle slave processors.

Algorithm		Number	Processors	
	4	8	16	32
Sync GA	3.5	6.9	12.7	23
Async iGA	3.96	7.94	15.93	31.96

Table 9: Communication Speedup Performance of Synchronous and Asynchronous GA on Modified Bump Problem

Table 9, shows the distributed speedup performance of the GA using both synchronous and asynchronous island implementations, when compared to a non-distributed synchronous GA. The results were based on the total time spent to calculate 100 generations, with a population size of 300 members. The iGA used 6 demes. For the synchronous scheme, idle process time arose due to poor synchronisation of the objective function calculation across the processor farm, as well as time spent waiting for the master processor to perform GA selection which includes CPU intensive clustering and vector sorting routines. The Asynchronous iGA implementation was successful in decoupling distributed communication from selection as reflected in the results. In both cases, speedup has not taken into account the extra processor needed for the master processor to GA calculation, hence only communication speedup is considered in this test.

The iGA implementation reduced significant idle time previously found on slave processors between generations. Residual traces of idle time found on slave processors was significantly small in comparison to the evaluation time of the objective function resulting in a net iteration efficiency of over 99% for 32 processors.

5.2.3 Cluster performance issues concerning dual boot implication

The creation of a Linux computational facility from resources intended for a different daily purpose using Windows NT was easily achieved through the use of a dual boot facility. Several issues were highlighted while implementing CFD problems using this strategy.

The dual boot feature poses an immediate barrier in gaining access to the computational resource when it is booted into the wrong operating system. Where the intention is to use the resource primarily for overnight computation, the individual processors can be configured to automatically reboot into the default Linux boot partition, at a designated time, which is the case used for this work. This strategy although very simple to implement, can cause problems both to NT users wishing to work during such times and computational resource users requiring access to Linux processors during the daytime. In this implementation, a small cluster of processors resides in a separate room to the rest of the processors. These processors reboot into Linux at a designated time, without causing significant disruption to users who are made aware in advance of this feature.

The second problem regarding the robust use of the facility for larger distributed processing using MPI, is when a processor is manually rebooted during computation. Without controlled management, such a reboot will cause the entire MPI process to crash on all associated processors losing all data. This problem is unavoidable since the primary use of the cluster is for NT based activities. The devastation of such an activity has been limited through encouragement of regular checkpointing of program data with a small associated computational cost as illustrated in Figure 39. Additionally, rebooting through the action of pressing CTRL-ALT-DEL on the keyboard has been trapped on all processors, sending an appropriate signal to all residing programs. On Linux, this signal is trapped in both the GA and Euler solver, and activates a final checkpointing procedure before a controlled exit of the program on all associated processors. A restart script residing on the master processor re-establishes a new set of available processors and attempts to continue the distributed computational job.

Some configuration time was required to establish the various processors required to increase the robustness and usability of the cluster when using a dual boot feature, however the process was straightforward and leads to substantial gains in the overall performance of the facility.

5.3 Robust marine CFD on a dual-use distributed network

The use of a Windows NT orientated network for Linux based computation can result in significant interruption to distributed jobs due to continuous rebooting of the processor by users into Windows NT during the day. For the majority of the year, most of the PCs are redundant (during a 24hr cycle every day of a year) and significant benefit can be offered for distributed computation. However a large risk of computation disruption exists during the day, which if unsupported would require a degree of human monitoring. In order to ensure a larger degree of stability during busy periods of PC usage, a control algorithm has been implemented. This allows individual processor shutdown, with a minimum user wait for access to the processor for teaching use while maintaining a high effective throughput. In addition, a rapid archive system using a tree XOR operation on the local distributed memory calculation state has been implemented to maximise throughput.

5.3.1 Teaching Load on Network

There are three main user groups of the Ship Science PC network:

- Formal Teaching sessions. Typically consisting of between 40 to 50 students, the timetabled sessions take place during 24 weeks and occupy 3 hour slots. In total of each term week, there are 5 to 6 such slots
- Individual taught Ship science students at present are of the order of 160 in total, evenly distributed between Years 1 to 4 of undergraduates and about 25 M.Sc. students. These students have access to the machines during office hours (8am to 6pm)for the 30 week academic year. A limited number of students (50) have access outside office hours.
- In addition to individual research student and staff. There are 40 postgraduate students, 20 staff, and visiting academics who have access to the computer room all year round.

Informal monitoring of usage, indicates that outside the formal sessions usage varies widely during the day and at weekends, but with typically a small number of individuals always requiring access 7 days a week but with access rare between midnight and 8 am. Based on the observed usage, it is estimated that the average annual load on each machine is only 10% for teaching purposes. The remaining processing time, if it can be accessed in a robust manner, offers considerable scope for use as a low-cost medium scale computational machine.

5.3.2 Failure Modes

The following important modes of possible failure which could affect the successful completion of a CFD calculation have been identified:

1. Power supply failure: Programmed outages occur 1 or 2 times a year with at least two weeks notice. Whereas, power supply interruption occurs no more than once every two years.
2. Network server failure (disk or machine): Again, a rare occurrence in an uncontrolled manner but deliberate shut down happens once every 1 or 2 months.
3. An individual machine or network connection failure.
4. Individual machine power off.
5. Request to use machine in teaching mode via ctrl-alt-del.
6. Lock up of MPI process when running on a given processor.
7. Insufficient memory space or processor capacity to continue calculation.

5.3.3 Consequence of CFD Calculation Failure

The computational cost of a calculation failure can be assessed in terms of:

1. The total amount of computational processing lost since the last secure record of the calculation state from which the calculation can be restarted. A trade-off needs to be made between the amount of time required to periodically save state and the risk of calculation failure.
2. Loss of available processing time. If a calculation is unable to restart automatically then this period could vary from minutes to days.

In each case a simple measure of the loss L, can be expressed as:

$$\mathrm{L} = C_L t_c N_p + t_r N_p \tag{86}$$

where C_L is the number of cycles lost since the last secure archive, t_c is the processor time per cycle, N_p the number of distributed machines in use, and t_r the time delay to restart.

5.3.4 Robust CFD Control Strategy

Any control needs to assure that the C_L is minimised and yet because of the primary demand of the machines for teaching within term time, any user requiring access must have a minimum user wait. Typically no more than a few seconds is deemed acceptable.

Having identified the possible failure modes a robust control strategy has been devised to ensure calculation stability. The objectives of the strategy can be identified as:

1. No. of lost cycles C_L is minimised for all failure modes.
2. The cost of archiving intermediate calculation states is minimised.
3. The maximum possibility of restart within a minimum t_r.
4. Minimum effort to encapsulate existing distributed applications.
5. Minimum of effect on teaching access to machines.

A specific strategy is identified for each of the six possible failure modes. Each control strategy relies on the use of a control process running in parallel with every calculation process on every processor.

5.3.5 Control process

The CTRL-ALT-DELETE action usually used by students to reboot the computer into NT has been trapped, and now runs a quick program 'controlnet'. The program checks local control files for computational resources using the robust control algorithms developed as part of this work. Essentially the Process IDentification number (PID) of any such computational programs are sent a signal similar to that of CTRL-Z. This signal is captured within the program and necessary data is saved directly to the file server for recovery, the program then proceeds to exit. Any other remote processors working in parallel with this program are sent a 'Hang up' signal via a remote shell. On capture, data is saved to local hard disc, and the program exits. The computer that received the reboot signal then proceeds to reboot.

This elaborate algorithm was necessary to ensure all information is saved within the smallest amount of time. By minimising network traffic to only one processor save (the rebooting proc.) and MPI shutdown, an acceptable network flushing time is achieved, which should not be of great inconvenience (user wait) to the user requesting NT service.

A recovery program (user supplied) is then initiated to piece together all files necessary to restart the computational process from the above last saved position.

5.3.6 Control Implementation on an Euler Solver

For uncontrolled failure (i.e. someone pressing the reset button or off switch) the control system relies on last saved data. Full data dumps within the Euler solution is costly requiring a total of $O(n)$ where n is the time required for n processors to send all flow variables to the master process where the server is used as a master, or $O(2n)$ if the saved data is sent to the server from master upon receipt.

Small intervals between data dumps are required if interruption is likely. If the intervals are too large, computational progress may not be achieved if interruption occurs before the next save. For the small interval times deemed necessary for reasonable progress to be offered, the cost of data dump is too great resulting in poor solution speedup through parallelisation (i.e. no gain in using the commodity network). To reduce the cost of data save, an intermittent strategy has been adopted similar to the 'RAID' process used for quick backup of network filestore's.

The RAID strategy creates an XOR checksum of all processor flow variables over a distributed PC architecture. The distributed RAID process is carried out by passing all flow variables to the master processor through a tree communication process. The basic schematics of the distributed RAID process implemented within the modified Euler solver is shown in Figure 40.

```
1. Receive $RAIDflag$ from $MASTER$.
2. Extract flow variables for backup and store in local array.
3. Send Sizeof flow variable array to $MASTER$.
4. Receive $MAXARRAYSIZE$ from $MASTER$ and allocate $RAIDArray[MAXARRAYSIZE]$.
5. Copy local flow variables into $RAIDArray$, fill remainder with 0.
6. $NPROC$ = number of $Processors$.
7. while $NPROC > 1$
    (a) If $myid < NPROC$
        i. If $myid >= RoundUp(NPROC/2)$
            A. Send $RAIDArray$ to $ProcessorID = myid - RoundUp(NPROC/2)$.
            B. Save local flow variable array to local disk as $Binary(double)$.
        ii. Elself $(myid - RoundUp(NPROC/2)) >= 0$ the
            A. Receive $Array$ from $Processor = myid + RoundUp(NPROC/2)$.
            B. $XOR\ Array$ with $RAIDArray$.
    $NPROC = RoundUp(NPROC/2)$
8. If $myid = MASTER$
    (a) Save $RAIDArray$ to network disc as $Binary(longlong)$
9. End.
```

Figure 40: XOR RAID Process

At each processor in the tree, all flow variables received are merged together through an XOR checksum such as $Checksum = 1\ XOR\ 2\ XOR\ 3\ XOR\ 4$ as illustrated in Figure 41. The time required for the merge operation is $O(Log_2(n))$ and therefore significantly more efficient than the full flow dump which will take $O(n)$ to pass all variables back to the master processor. The final checksum is saved to server as a compact binary file, and each processor saves its own flow variables to local disc. In the event of uncontrolled processor failure, information lost can be recovered by passing the remaining processors locally saved data, back through the XOR checksum such as $B = Checksum\ XOR\ 1\ XOR\ 3\ XOR\ 4$.

Recovery from a RAID backup is implemented as a separate process, so that it does not interfere with the solver itself. The recovery algorithm is shown in Figure 42, which must be distributed across all remaining processors except that which has failed. A simple UNIX shell script has been used to create a hosts file containing all remaining processors, and to identify the missing failed processor id which is passed to the distributed recovery process. The recovery process itself, recovers all processors flow variables to network filesystem, from where it can be re-distributed

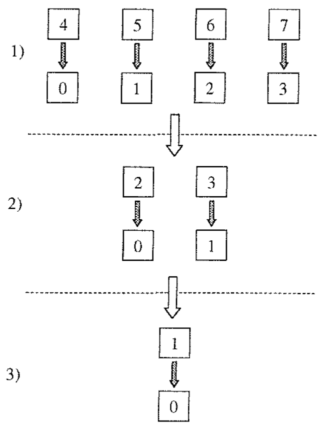

Figure 41: Distributed XOR RAID Communication Process between 8 Processors

back to the designated processor id's during the flow solver startup process. If too many failures occur, the missing flow data cannot be recovered from this simple XOR implementation, and the computation will be forced to revert back to the last full data dump.

The speedup loss in performing the XOR Merge Tree at regular intervals of 5 Euler iterations plus an additional full data backup every 20 iterations, is shown in Figure 43 which was measured over 1000 iterative cycles in total, each consisting of four pseudo Runge-Kutta time iterations. The performance of a full data save every 5 iterations is also shown for comparison. The recovery in speedup loss is significant, and the burden of incorporating the operation at such frequency required for unstable networks is justifiable, especially over medium distributed networks. The total time to recover a single lost processor from 16 PCs, was $2.6s$ compared to $8.2s$ to redistribute last saved data to all processors again. For larger networks, more sophisticated checksum operations could be used to reduce the expense as well as to provide immediate backup for the loss of more than one processor.

```
1. Determine Failed Processor.
2. Start MPI Recover Process across all remaining processes and assign $MASTER = processorid(0)$.
3. Load up $Binary(double)$ flow variables from local disk to $Array$.
4. Send $Size(Array)$ to $MASTER$.
5. Receive $MAXARRAYSIZE$ from $MASTER$ and allocate $RAIDArray[MAXARRAYSIZE]$.
6. Copy local flow variables into $RAIDArray$, fill remainder with 0.
    (a) If $myid > MASTER$
        i. Send $RAIDArray$ to $MASTER$.
    (b) Elseif $MASTER$
        i. Load $Binary(longlong)$ $Array$ from network disk.
        ii. $XOR$ $Array$ with $RAIDArray$.
        iii. For $i = 1$ to $i < NPROC$.
            A. Receive $Array$ from $Any$.
            B. Load Previous Input Data for Received Flow Info.
            C. Backup Previous Data.
            D. Overwrite with Received Flow Info and save as Current Input Data.
            E. $XOR$ $Array$ with $RAIDArray$.
        iv. Load Previous Input Data for Missing Flow Info.
        v. Backup Previous Data.
        vi. Overwrite with $RAIDArray$ Flow Info and save as Current Input Data.
7. End.
```

Figure 42: XOR RAID Recovery Process

5.3.7 Control Process for Genetic Algorithm Search with Panel Solver

A simple strategy is implemented to control processor interruptions during GA search. On completion of each member evaluation, the result is returned to the Master process where it is appended to a file on the server. In the event of processor interruption of any type, the 'controlnet' program is activated which informs all other processors of the shutdown status. All objective function calculations are aborted immediately so that a quick shutdown time is acquired. Once the Master processor has returned to console status, the GA is restarted with a continuation flag. The GA loads the population and convergence status details required and sends any members that have yet to be evaluated to the remote slaves so that they can restart their evaluation.

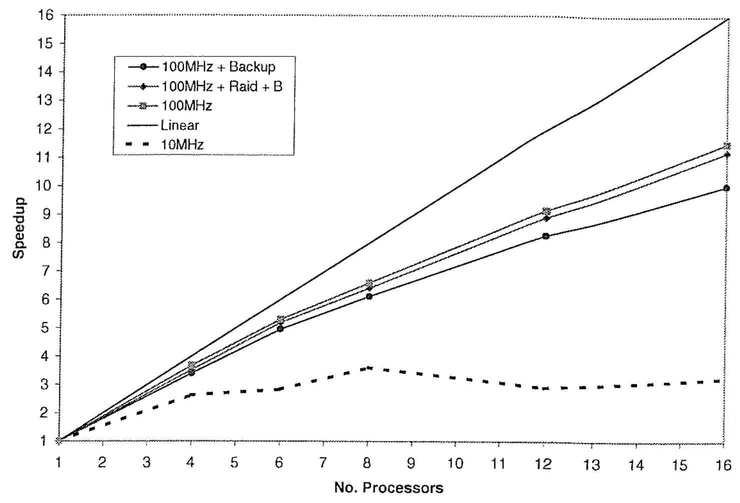

Figure 43: Speedup Performance for Euler Solver Using Regular RAID Tree Merge of Local Data

5.4 Summary

A commodity-processing network has been established from existing resources without conflicting with their original usage intent. Almost linear speedup performance has been achieved for the parallel computation of over 300,000 objective function calculations when using a Genetic Algorithm for design optimisation. Good speedup was measured during the computation of an Euler solver based on an unstructured grid for up to 16 processors. The use of dual boot can significantly affect the robustness of the computation facility, and some progress was made in this implementation to reduce this effect. Overall, an excellent computational facility can be harnessed from existing PC resources at little additional cost.

Chapter 6

Evaluation of airfoil shape parameterisation Using a Genetic Algorithm

In order to examine how different geometric representation schemes can be used with a GA, an airfoil optimisation study has been conducted on several objective functions. A variety of parameterisation techniques have been used with CFD orientated search, within the literature. Samereh [55] generalised geometry parameterisation techniques into the following categories: basis vector, domain element approach, partial differential equations, discrete, polynomial and spline, CAD-Based, analytical and free-form deformation (FFD). The suitability of these methods is based upon the efficiency, effectiveness, ease of implementation, and scope for global and perturbation search.

Two techniques of particular interest were those based on spline techniques which are the most popular representation used by the research community, and the analytical function approach. To analysis their practical use to airfoil optimisation, the viscous coupled 2D panel solver XFoil [67] has been used to judge the fitness of candidate airfoil sections based on design point performance.

6.1 Evaluating the Performance of Airfoil Sections Suitable for Ducted Thruster Units

For this design problem, an airfoil section is sought for use in a thruster unit suitable for deep sea Tethered Unmanned Underwater Vehicles (TUUV). These units commonly used for operations such as oil rig maintenance, operate mainly at depths where cavitation will not occur based on the pressure exerted on the foil.

Ducted thruster units offer several efficiency advantages over traditional propeller arrangements by offering a lower propeller loading (the loading is offset by using the duct to draw a larger volume of water into the propeller), and additional thrust due to the acceleration of flow over the duct. An additional advantage offered is that the duct wall proximity to the blades reduces the effects of tip vortexes, particulary strong due to the finite span of the propeller.

In an attempt to eliminate this vortex drag, a ring propeller arrangement has been proposed where the propeller is mounted to a thin ring that sits flush within the external duct. This arrangement tends to lack the efficiency offered by traditional ducted propeller units using a much simpler arrangement with a small gap between the propeller and duct to reduce tip vortexes. The main drawbacks to the ring propeller arrangement are mainly the mechanical problems of power transmission, seals and centrifugal bearing problems associated with the new designs.

Remote Operated Vehicles (ROVs), or Tethered Unmanned Underwater Vehicles (TUUVs), require propulsion units that offer high efficiency while relatively light weight and good flexibility for the positioning of the thrusters. Such units currently use small electrical units driving the shaft of the propeller, and are mounted from the nozzle using a 'spider' type bracket. However one drawback to this arrangement is that the position of the motor disrupts the flow of water into the propeller.

A natural progression, drawn from the basis of ducted ring-propellers, has been the idea of tip-driven propellers (TDPs). This concept involves having either a mechanical drive or electrical motor encased within the nozzle walls. The use of an electromagnetic drive with the absence of physical contact between the drive system and propulsor, offers significant advantages to the mechanical layout which still involves issues such as the seals problems associated with the mechanical drive.

The hydrodynamic performance of a TDP was investigated by Hughes [66] for a unit using a highly efficient permanent magnetic motor design. The efficiency performance still lagged that of traditional ducted units and it was proposed that the unit should be hydrodynamically optimised to examine whether such performance could be recovered. One of the non-optimised and novel features of the tested unit, was the bi-directional characteristics of the TDP, offered to enhance positioning efficiency and effectiveness required by TUUVs.

The hydrodynamic optimisation of the bi-directional tip driven thruster unit using the Geometric GA will be investigated in this chapter.

The two dimensional performance of airfoil sections will be examined for low lift coefficients of 0.3 and 0.6, frequently found with propeller designs to enhance efficiency by reducing the propeller load. Single point optimisation using the Geometric GA on fixed 2D topologies, is currently considered at a similar Reynolds number as that of the tested propeller section at 70% radius.

The investigation will initially focus on the parameterisation of traditional airfoil sections for the above design problem. Successful solutions will then be tried on the bi-directional problem investigated by Hughes.

6.1.1 The Cost Function

The objective of both airfoil problems is to minimise the drag coefficient for a given lift coefficient, and Reynolds number.

$$Objective = Min(C_D) \tag{87}$$

subject to $C_L = defined$ and $Re = 0.95 \times 10^6$

To ensure that practical designs are found several geometric constraints are used such that

$$\begin{aligned}
&y_{upper}(x) > y_{lower}(x) : \text{for } 0 < x < 1 \\
&y_{upper}(0) = y_{lower}(0) = 0.0 \\
&y_{upper}(1.0) = -y_{lower}(1.0) = -0.0025 \\
&t_{x=0.25} > 6\% \text{ chord} \\
&|y_{upper}(x) - y_{lower}(x)| > 2\% \text{ chord for } 0.05 < x < 0.9
\end{aligned} \tag{88}$$

An additional constraint was also placed on the pitching moment such that

$$-0.1 < C_m < 0.02 \tag{89}$$

To accommodate these constraints, the dynamic penalty scheme given in Chapter 4 is used, with equality constraints treated as inequality constraints to ease the stochastic search process. The construction of the penalty scheme is orientated to provide a objective function of which a minimum is to be found. The final form of the objective function is given as:

$$\begin{aligned}
&U = C_{D_{total}} \\
\\
&c_1 = 10(C_L - C_{L_{required}}) \\
&c_2 = 100(MIN(\delta y_{0<x<1}) - 0.005) \\
&c_3 = 100(t_{x=0.25} - 0.06 \times chord) \\
&c_4 = 100(MIN(\delta y_{0.05<x<0.9}) - 0.02 \times chord) \\
&c_5 = 10(C_M + 0.1) \\
&c_6 = 10(0.02 - C_M)
\end{aligned} \tag{90}$$

6.1.2 The Genetic Algorithm

To provide a robust analysis of geometric parameterisation techniques, the distributed asynchronous iGA with sharing, will be used as detailed in Section 4.3.1. Such an algorithm allows some facilitation of niching control through both the island population scheme as well as the use of resource sharing, thus allowing several good solutions to propagate through the evolution process without the risk of population takeover. The solution quality gain from using niching control has been highlighted by Doorly [52] through the use of iGA's, and the rugged surfaces illustrated by Holden [2].

One consequence of using both island and sharing schemes, is that larger populations are required to support multiple niche formation. For 20 parameter optimisation, a population size of 200 will be used with 50 members per deme. The use of only four demes will provide the computational performance gain of adopting asynchronous migration, while providing enough population resources for each deme to support between 8-15 niches.

Linear Rank scaling with a selection intensity of 1.6 is used with Elitist Stochastic Universal Selection within each deme. Two point crossover with a rate of 85% is used, combined with a mutation rate of 2%. All simulations will be initialised from a base foil section, allowing 15% random perturbation of the design space to form generation zero.

6.1.3 Issues Concerning the Coupling of XFoil with the GA

Problems encountered when running XFoil from the Objective Function, highlighted several problems when semi-random airfoils are considered. Main problems highlighted, includes: XFoil program crash, excessive non-convergent calculation cycles, permanent program freeze until user intervention (via use of system kill command), excessive or inadequate panel distributions, non-converged boundary layer solutions, divide by zero segmentation faults, and infinite numbers being returned.

For the majority or problems, the use of constraint analysis could be used. To assist in the basic running of XFoil, airfoil pre-constraints were analysed to impose a feasible section condition. This ensured that only true sections with a positive volume (i.e. the upper surface remained above the lower surface for all points along the x-axis), and that either minimum or maximum thickness requirements were met. These were imposed as hard penalty functions that simply ensured that the objective function returned a poor fitness result with appropriate penalty violations without using XFoil itself.

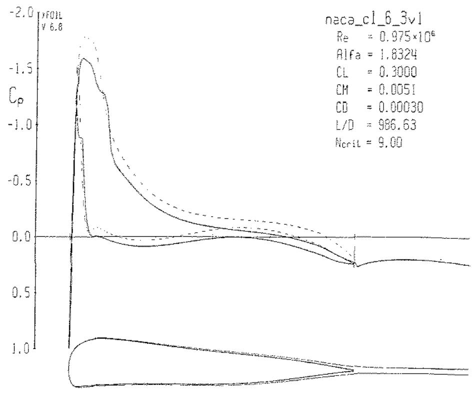

Figure 44: A Misleading Solution that Appears as a Good Solution

Careful post-processing of the XFoil log file, passed separation or non convergent problems to the appropriate penalty functions. Other penalty conditions used, ensured that key performance parameters were met. Only inequality constraints were used with the fixing of lift coefficient for all runs to avoid use of an equality condition. Earlier attempts of introducing equality conditions resulted in very poor GA convergence, thought to be a problem of using such conditions with stochastic processes.

Despite the use of geometrical and performance constraints, several of the problems discussed above hampered productive use of XFoil with GA search. Source code modification was finally used to prevent excessive calculation cycle runs. In addition to this, a separate program was run at GA startup that monitored the process status of XFoil runs, sending any sleeping or long process runs the UNIX SIGINT signal if required, so that the GA could continue with the next candidate.

Several problems are still emerging during the GA search process such as misleading results as shown in figure 44, with an incredible drag coefficient, or implementation of penalty conditions demonstrated in figure 45 which returned a very good drag result, but is almost impossible to manufacture. The use of niching with the GA ensures that the rest of the population can remain generally unaffected by any misleading results.

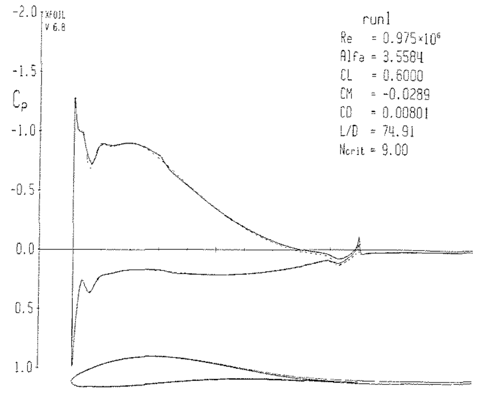

Figure 45: Poor Constraint Consideration May Lead to Unpractical Optima's

6.2 Spline Representation Techniques

Spline interpolation techniques [50] are readily used by engineers through Computer Aided Design. Such interpolation techniques appear frequently in the literature, popular for their generality and ease of implementation. Bezier, BSpline and NURBS are such representations that have been used for optimisation, using the control points as design variables.

6.2.1 B-Spline Parameterisation

B-Spline's are commonly adopted in CAD systems to define smooth curve shapes, and have shown some success in their application to automated design[35][36][37]. B-Spline curves are interpolated from a set of control points which are generally defined as coordinate sets $\bar{\mathbf{P}}_i(x, y)$. The B-Spline curve is then given as Equation 91

$$\bar{\mathbf{R}}_g(u) = \sum_{i=1}^{n} \bar{\mathbf{P}}_i N_{i,p}(u) \tag{91}$$

where $N_{i,p}(u)$ are the Bernstein basis functions of p-degree $(orderp + 1)$ defined as

$$N_{i,0}(u) = \begin{cases} 1 & if\ u_i \leq u \leq u_{i+1} \\ 0 & otherwise \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \qquad (92)$$

where u_i are called knots. The tightness of the curve to a given control point can be controlled by the form of *knot* vector given, whereas the number of control points that affect the position of a point is controlled by the *order* of the curve. Guidance to the form of the knots is given in [50], however only uniform knot vectors are considered in this study such as $U(0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1)$ for a 2^{nd} order curve with five control points. Of course there is no reason why parameterisation of the knot vector itself could not be part of the overall design vector to be optimised.

To reduce the number of design variables considered, the control points were fixed in the chordwise positions (x), allowing freedom in the y direction to define section curvature. A total of 20 poles were used as design variables, with leading edge and trailing edge poles fixed.

For the most basic of implementations, a 'delta' parameterisation was used where pole movement was defined as a percentage of a given position. In this test the given position was defined by a NACA0012 section [104]. The extent of parameterisation for 50% perturbation is shown in Figure 47, along with the resultant section found from a single pass with the GA using a population size of 200 members. It is clear that such a parameterisation is too restrictive for GA search (since GA's are generally used to find a near global optima rather than a local optima), and therefore a full parameterisation of the BSpline method has been attempted. The full range of sections offered by the scheme is shown in Figure 46, with the optimised section compared to the perturbation results demonstrated in Figure 48.

One problem that most of these interpolation techniques exhibit, is the tendency to produce 'wiggly' shapes. Figure 49 illustrates the problem more predominantly with regards to the pressure distribution. Reuther and Jameson [35] observed such problems and proposed the use of a low-pass filter to smooth the curve. The use of filtering is expected to introduce redundancy into the parameterisation. Careful examination of the leading edge of the resultant design reveals a poorly defined nose. The definition of the leading edge could be significantly improved by imposing tangency. This is easily achieved through the use of two spline sections for the upper and lower surface

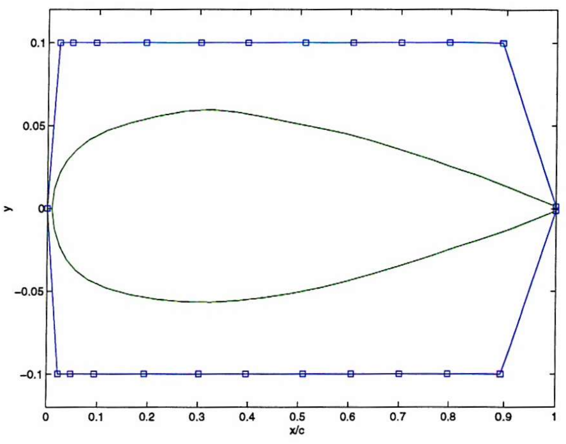

Figure 46: Limits used for B-spline representation with resulting section

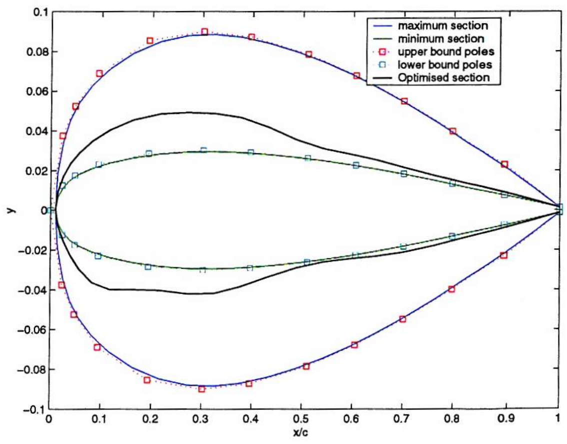

Figure 47: Optimised b-spline section by perturbating ordinals about a reference section

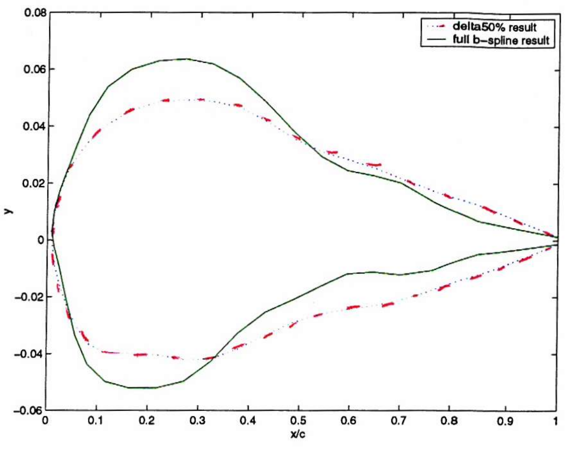

Figure 48: Resultant best sections from implementations of full and delta parameterisations

with the second control point fixed at x=0 but allowed to move upwards. Lepine and Trapanier [37] obtained practical smooth sections using BSpline and NURBS representations. In their implementations, the control points were not restricted to movement in one dimension which is thought to cause excessive wiggles in surfaces through inefficient bunching of the Bernstein interpolation functions. However their representation required over 40 design variables to define a simple airfoil section.

Increased Freedom of BSpline Pole Movement

By allowing the BSpline poles to adapt in both x and y directions, much smoother shapes were achieved using 16 poles which required 32 design points to be optimised. Figure 50 shows the resultant section and defining BSpline poles, when allowed to move freely. The pressure distribution for the optimised section given in Figure 51, is a significant improvement to that of Figure 49.

6.2.2 Bezier Representations

The Bezier curve is a special non-piecewise form of the B-Spline which can be used to produce smooth curves providing that the order of the curve is kept low, as higher orders can lead to oscillations in the shape. The Bezier interpolation of control points

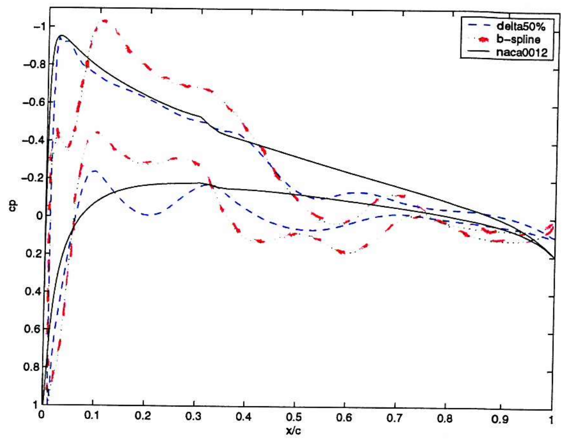

Figure 49: Comparison of optimised pressure distributions

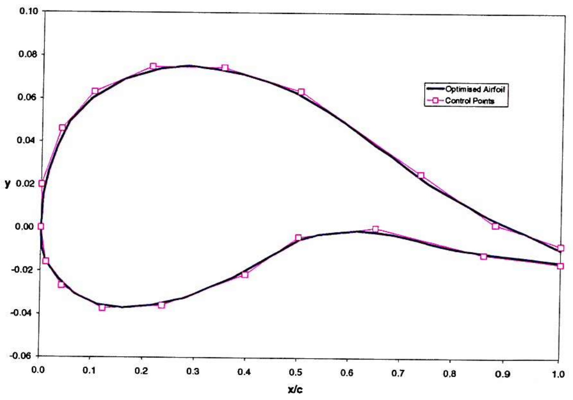

Figure 50: Resultant section when allowing BSpline poles to move in x and y directions

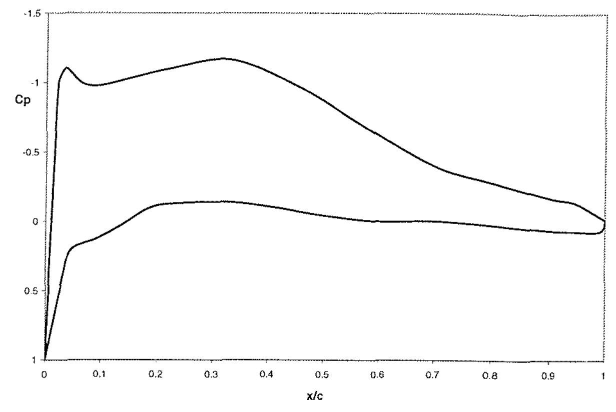

Figure 51: Pressure distribution attained by allowing BSpline poles to move in x and y directions

$\bar{\mathbf{P}}_i$ is defined as

$$\bar{\mathbf{R}}_g(u) = \sum_{i=1}^{n} \bar{\mathbf{P}}_i \mathbf{B}_{i,n}(u) \tag{93}$$

where the basis functions $\mathbf{B}_{i,n}(u)$ are the classical n^{th} degree Bernstein polynomials given in Equation 94

$$\mathbf{B}_{i,n}(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i} \tag{94}$$

The use of Bezier curves in shape optimisation has successfully been used by Holden [2], Reuther and Jameson [35], Lépanier [37] and Doorly [34], producing smooth sections without the oscillatory pressure distribution characteristics seen in Section 6.2.1. This method has been investigated due to its ability to produce smooth geometries without the need to introduce more design vectors that are required by B-Spline methods to reduce oscillation. Once again, 20 moveable control points are used to represent the section shape, with movement restricted to the vertical direction only.

The resultant section produced for the $C_L = 0.6$ test case is shown in Figure 52. The section produced is significantly smoother than the equivalent B-Spline result, an observation echoed by Holden [2] and Reuther and Jameson [35]. It should be

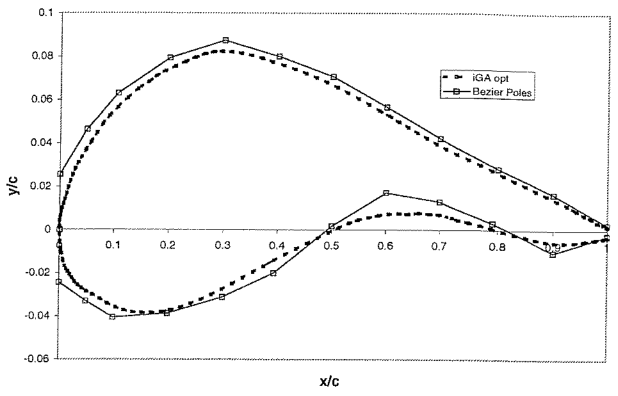

Figure 52: The resultant Optimised Section from a Bezier Spline Representation

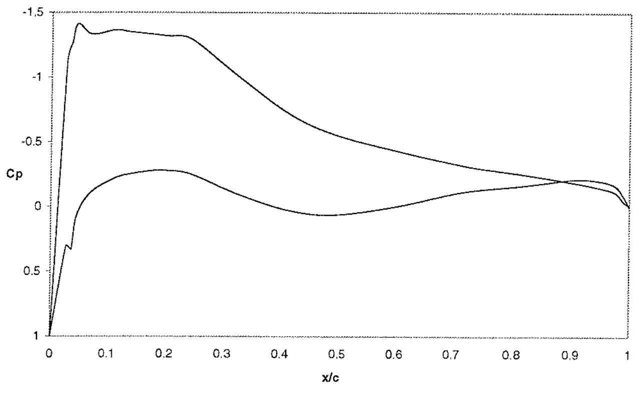

Figure 53: Pressure distribution obtained using the Bezier representation ($C_L = 0.6$)

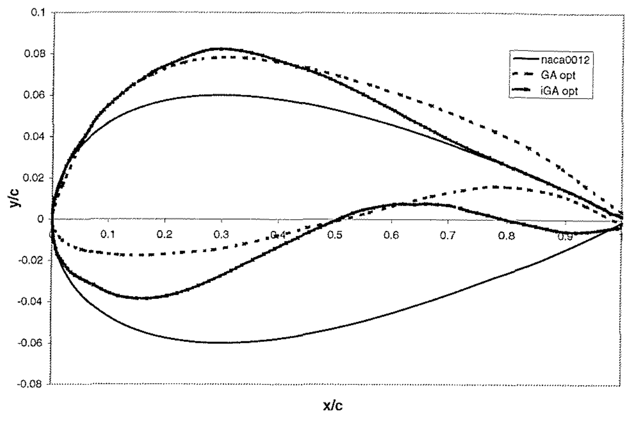

Figure 54: Comparison of optimised Bezier sections obtained using an iGA ($C_L = 0.6$)

noted that this representation may not be able to capture the detail that can be discovered by B-Spline methods, due to the high order implementation of the Bernstein interpolation functions. Bezier methods would therefore make an excellent representation technique for initial searches of the design space. Full B-Spline or even NURBS curves can then be used to continue the search, either with GAs or higher fidelity hill-climbing search methods such as Powells Direction Set Method [105].

To investigate the effect of switching off the asynchronous island scheme on the quality of the result found, the previous problem was tried on a niching GA with a population size of 200. A slightly different section was obtained, and is compared to the iGA result in Figure 54. The convergence to the two methods in Figure 55 shows that although different section shapes were obtained, in objective terms they both represented different local optima of similar quality. The robustness and repeatability of results from the same representation is still an ongoing problem.

6.3 Analytical Functions

Hicks and Henne [41] introduced a compact formulation for the parameterisation of airfoil sections . This method is based on adding sets of sine bump functions linearly to an existing baseline airfoil section. The amount of each bump added to the baseline

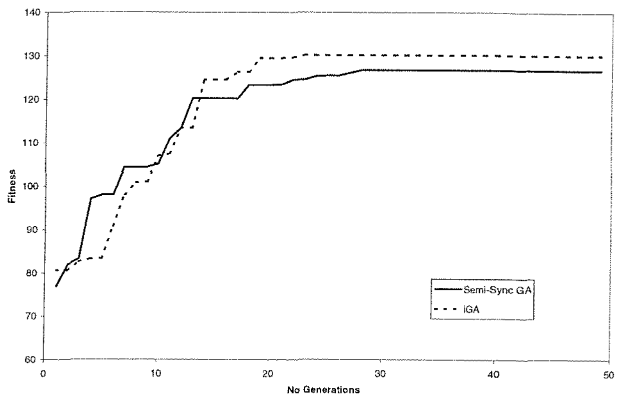

Figure 55: Superior convergence obtained through island Genetic Algorithm

section is defined by a coefficient (design variable). If all the coefficients are set to zero, the resultant gives the base section. This method was implemented by Reuther and Jameson [35] using 25 functions, acquiring much smoother sections than those found based on spline representations. Hager et al. [42] and Elliott and Peraire [44] used 10 shape coefficients based on different shape functions.

The value of shape functions can be enhanced by finding an orthogonal set. The use of ortho-normalised shape functions in the representation of airfoil sections was adopted by Kuruvila et al. [46] based on the NACA four series. Chueng [47] and Drela [48] also demonstrated the effectiveness of this method based on a sine series.

Chang, Torres, and Tung [106] demonstrated the parameter efficiency of ortho-normalising base functions, recovering a NACA0012 section to an acceptable error with just the first four modes, and with eight modes to completely recover a transonic section. To evaluate the parameter efficiency of ortho-normalised modes further, the first four modes of ortho-normalised shape functions are to be considered in the aerofoil optimisation problem.

The base functions used by Kiruvila et al. are given in Equation 95, as

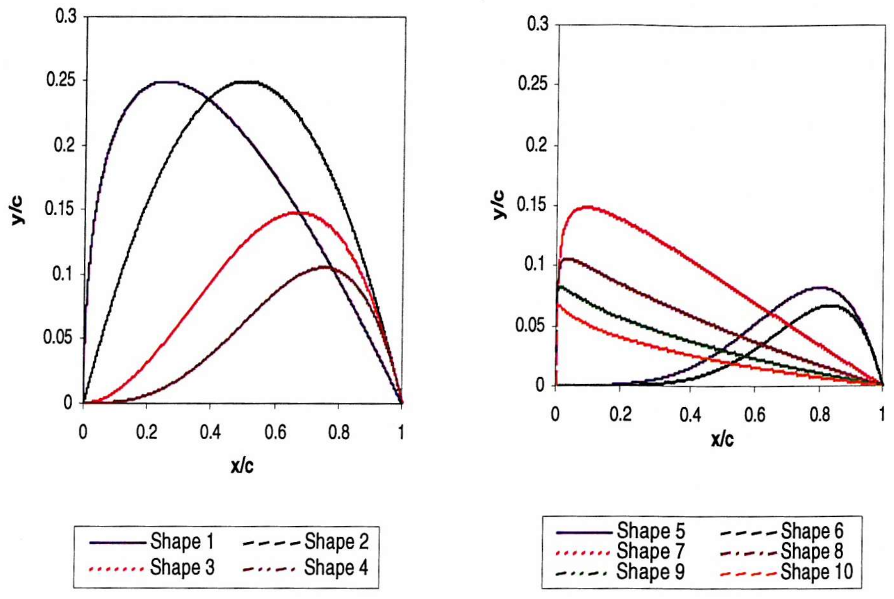

Figure 56: NACA Series Shape Functions

$$
\begin{array}{ll}
 & y_5(x) = x^4(1-x) \\
y_1(x) = 1 - x & y_6(x) = x^5(1-x) \\
y_2(x) = x(1-x) & y_7(x) = \sqrt[3]{x} - \sqrt{x} \\
y_3(x) = x^2(1-x) & y_8(x) = \sqrt[4]{x} - \sqrt[3]{x} \\
y_4(x) = x^3(1-x) & y_9(x) = \sqrt[5]{x} - \sqrt[4]{x} \\
 & y_{10}(x) = \sqrt[6]{x} - \sqrt[5]{x}
\end{array}
\tag{95}
$$

and plotted out in Figure 56.

The ortho-normalised modes that are to become the parameterisation for the problem were obtained by Gram Schmidt ortho-normalisation process and are shown in figure 57. The Gram Schmidt process used is detailed in Appendix C.

The final aerofoil shape is obtained by adding each normalised function as a weighted sum such as:

$$
\bar{\mathbf{R}}_g(x) = \sum_{i=1}^{n} a_i g_i(x) \tag{96}
$$

where a_i are the design variables to be optimised, and $g_i(x)$ are the ortho-normalised

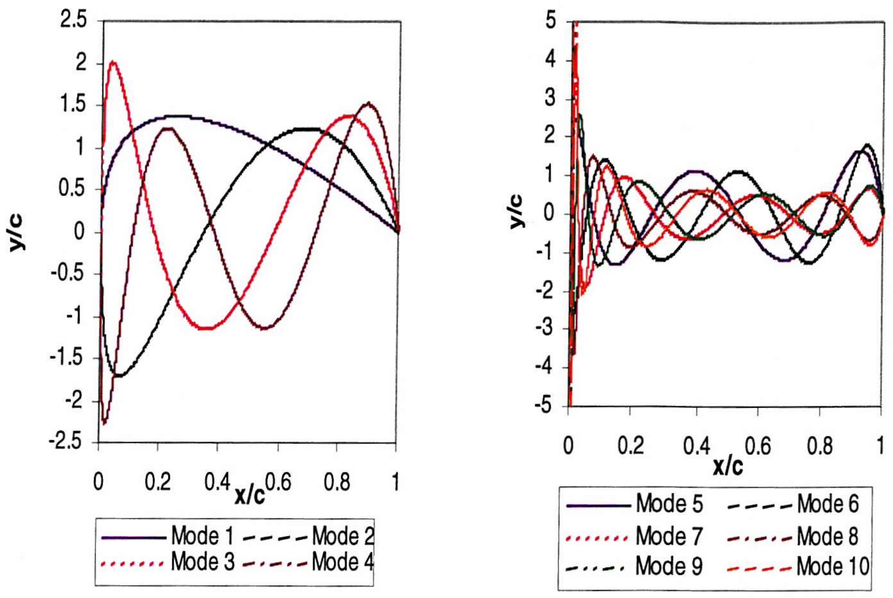

Figure 57: Ortho-normalised modes of NACA Series Shape Functions

functions of y_i which are the base functions given in Equation 95.

In the application of the orthogonal modes to an airfoil design optimisation problem, the first four modes were used for each surface, resulting in only eight design parameters. The resultant solution for the $C_L = 0.3$ design point is shown in Figure 58. This section obtained a far superior objective solution, to that of the Bezier solution, with a much smoother and useable section. Examination of the parameterisation of the delta b-spline method in Figure 59, reveals that such a section lies outside its bounds, highlighting the limitations of the delta approach for global optimisation. The pressure profile of the resultant section shown in Figure 60, has changed significantly from the NACA0012 section. A sharp join is noticeable at the leading edge of the foil. Since there is no continuity of curvature properties in the join between the upper and lower surfaces, it is not surprising to find these features common in such implementations as also observed by [46].

In an attempt to impose tangency at the leading edge, BSpline curves were used for the upper and lower surface with an additional control point placed above the leading edge point imposing the tangency condition. Ortho-normalised functions were used as the design variables, with the resultant profile used to define the fourth order

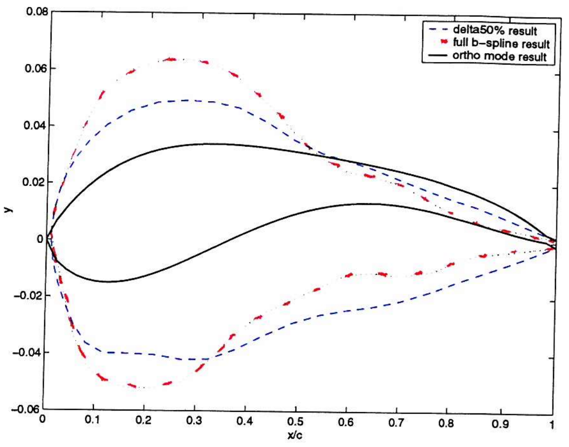

Figure 58: Resultant sections obtained using orthogonal functions

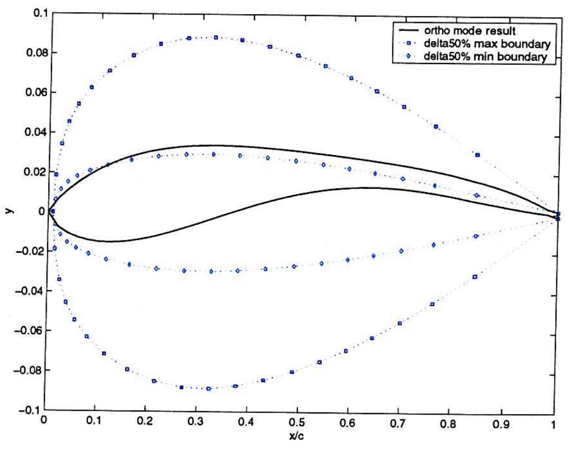

Figure 59: Optimal section of the orthogonal representation exceeds the delta b-spline boundaries

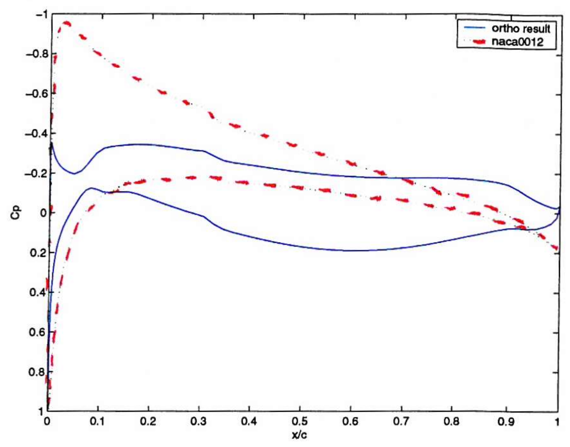

Figure 60: Optimised pressure distribution from orthogonal representation

BSpline control points. For the first design case investigating airfoil sections at a low lift coefficient, which is often required in propeller sections, the resultant airfoil section shown in Figure 61, was significantly improved. A large similarity is evident despite the improved leading edge condition between the pressure distributions shown in Figures 61 and 59. At a much higher lift coefficient, the rearward loading found in the previous case study was abandoned for a more traditional leading edge loaded section.

The current set of orthogonal modes can be applied to a new Bi-Directional airfoil section proposed by Huges [66] for a thruster unit with equal bi-directional thrust performance, suitable for deep sea Tethered Unmanned Underwater Vehicles (TUUV). Hughes used the upper surface of a Kaplan K4-70 [107] airfoil section to construct the bi-directional airfoil, with the upper edge rotated 180 degrees to form the lower edge.

To optimise such sections, the same principle was applied in the parameterisation based on the first four ortho-normalised modes, using BSpline control points imposing tangency conditions at both leading and trailing edge. The final parameterisation required ten design variables in total. The optimised sections shown in Figures 63 and 64 for lift coefficients of 0.3 and 0.6 respectively, were comparable in performance to a NACA0012 section in forward direction only. This is a significant improvement to the

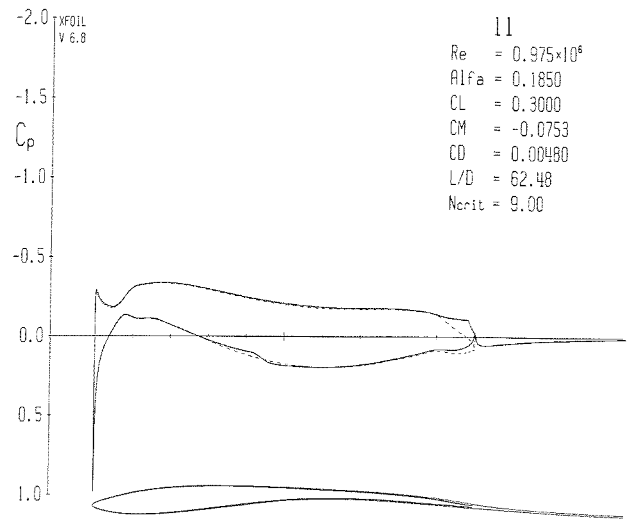

Figure 61: CL = 0.3 Result with Tangential Geometric Condition Applied to Leading Edge

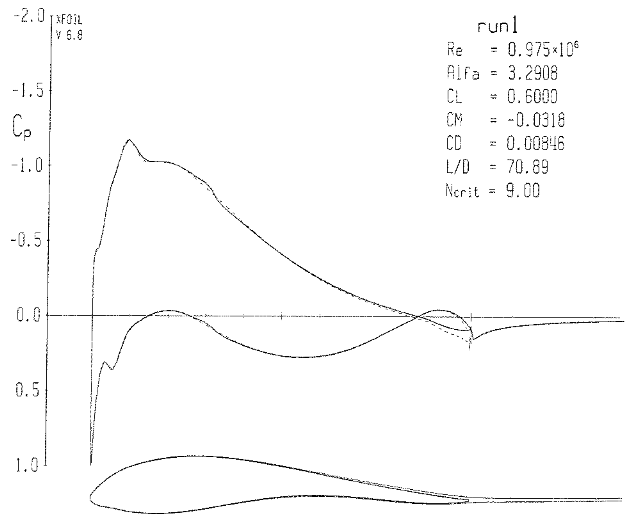

Figure 62: CL = 0.6 Result with Tangential Geometric Condition Applied to Leading Edge

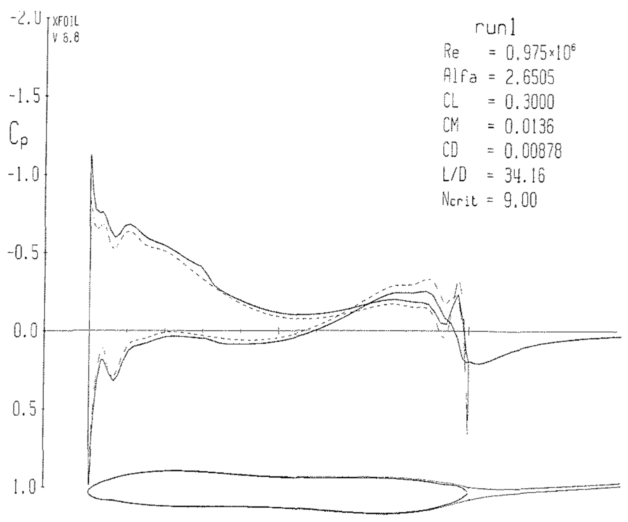

Figure 63: Solution of a Foil Suitable for the Bi-Directional Operation of a TUUV Thruster Unit, $C_L = 0.3$

section used by Hughes shown in Figure 65. The drag polar of the three airfoil sections given in Figure 66, illustrate this improvement. The 'S' shape feature of the optimised sections produce a flat polar characteristic over a wide range of lift coefficients. This performance characteristic would be significantly advantageous to a manoeuvring vehicle as there would be little performance cost associated with operating away from the design point. One of the disadvantages of basing optimisation on a single design point is featured in the $C_L = 0.3$ polar, where the drag polar increases drastically about $C_L = 0.6$. A combined objective may be more suitable for a final design study in order to provide performance strengths that include both design points.

6.4 Efficient Parameterisation Through Ortho-Normalised Aero-Functions

Although some success was made in obtaining a smooth leading edge when ortho-normalised functions are applied to airfoil design, a further opportunity to reduce parameterisation potentially exists through the ortho-normalisation of functions that

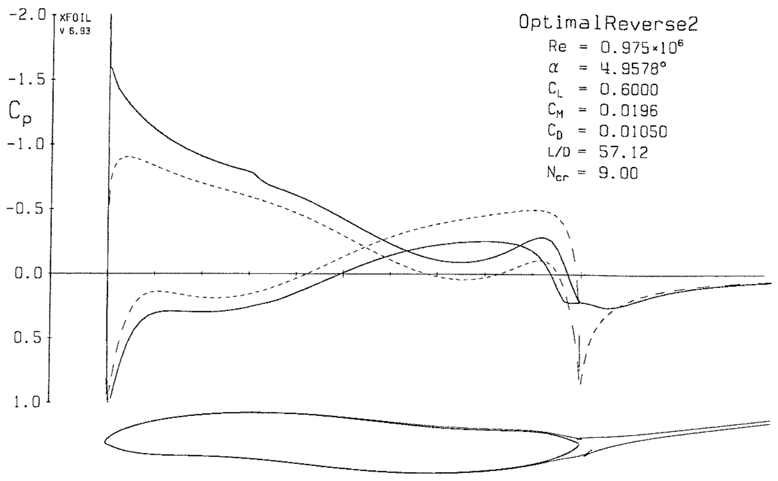

Figure 64: Solution of a Foil Suitable for the Bi-Directional Operation of a TUUV Thruster Unit, $C_L = 0.6$

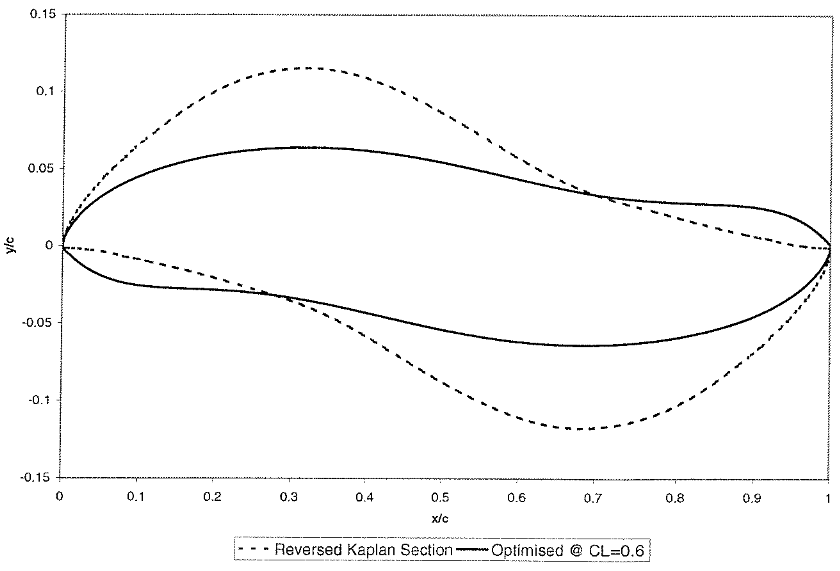

Figure 65: Original Proposed Section formed through a rotated Kaplan K4-70 Section

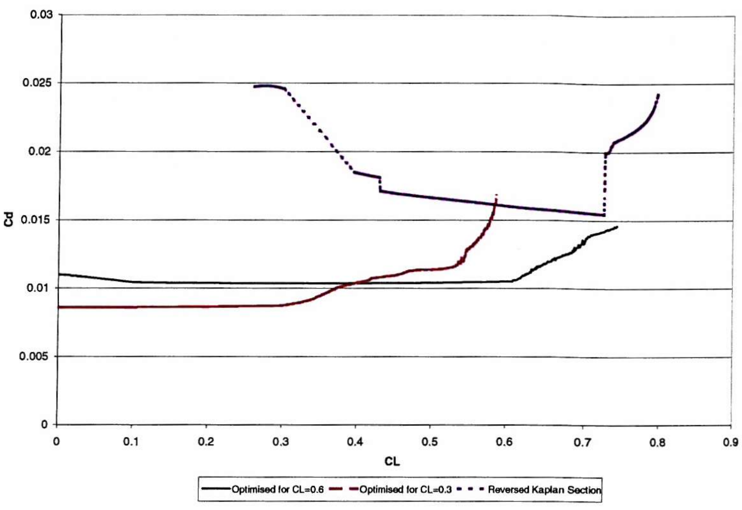

Figure 66: Drag Polar Comparison of Optimised Sections for Reversible Section

define the entire airfoil section instead of just the upper or lower surfaces.

6.4.1 Construction of the orthogonal set

In order to explore this idea, an ortho-normalisation process has been implemented for the reduction of a discrete set of curves. Instead of using mathematical functions, existing aerofoils are used to form the basis set. Gram Schmidt ortho-normalisation is applied to each airfoil which are treated as discrete curves. Twenty base airfoil candidates were chosen from over 1000 airfoil sections available in [108], for the wide contrast in shapes offered between them. The airfoil basis sections are shown in Figures 67 and 68.

In Gram Schmidt ortho-normalisation, basis functions must cover the range $0 \leq x \leq 1$, so the upper surface is rotated about the $Y - axis$ from the LE, and the new curve scaled and translated to satisfy the required range. The resultant base 'Aerofunctions' are shown in Figures 69 and 70. The effectiveness of the normalisation process is enhanced by ordering the functions in terms of maximal differences between each of them. The distance between two functions can be found from

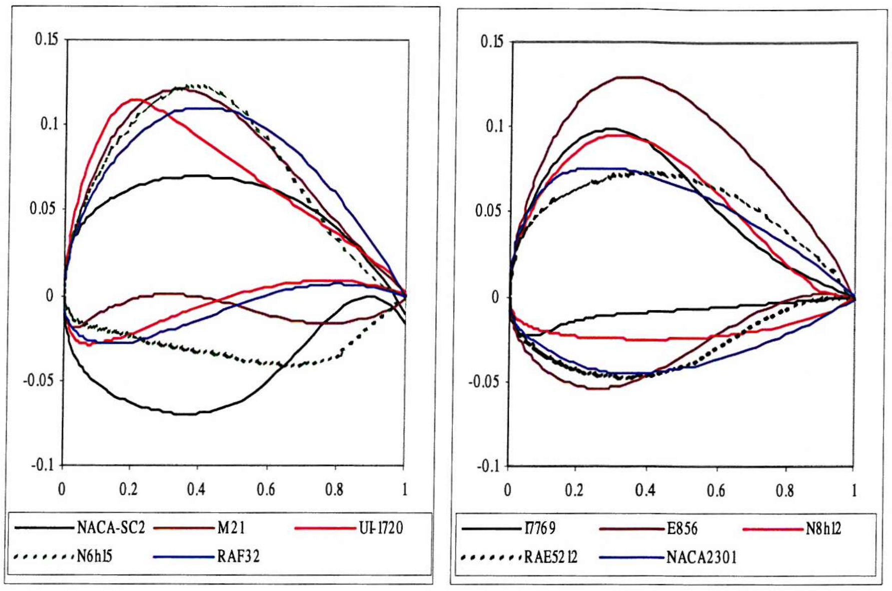

Figure 67: Basis Airfoils for Constructing the Orthogonal Set

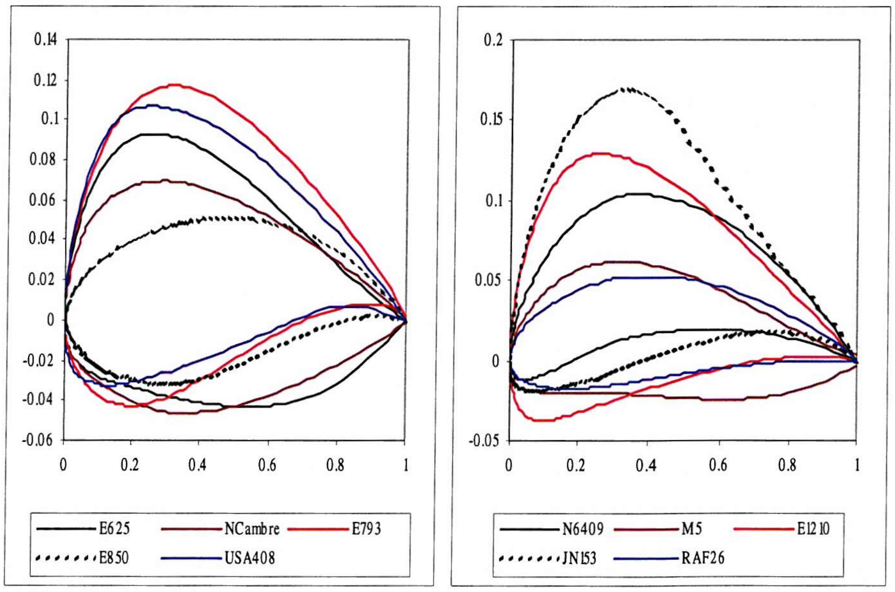

Figure 68: Additional Basis Airfoils for Constructing the Orthogonal Set

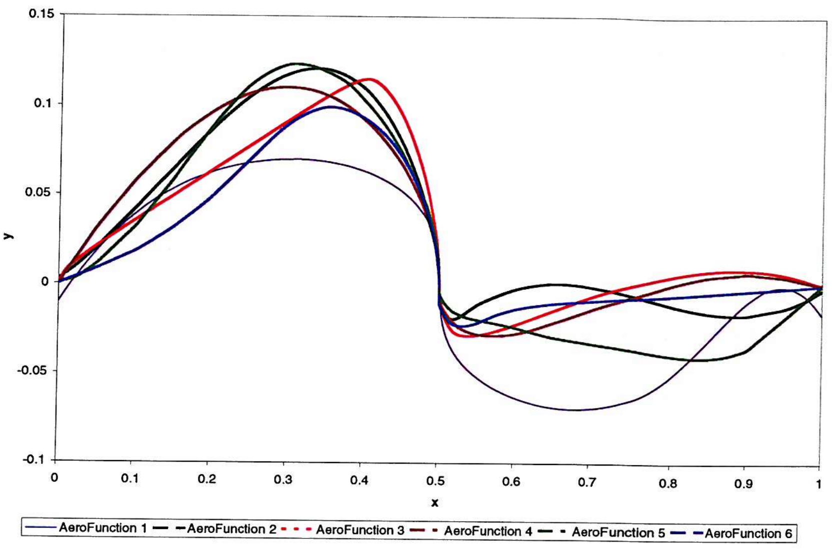

Figure 69: Basis Functions for Gram Schmidt Process

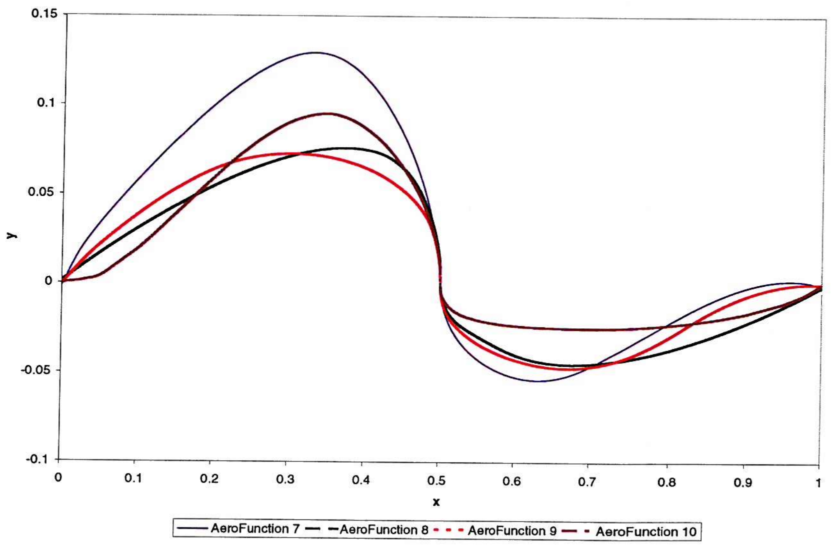

Figure 70: Basis Functions from Additional Basis Airfoils

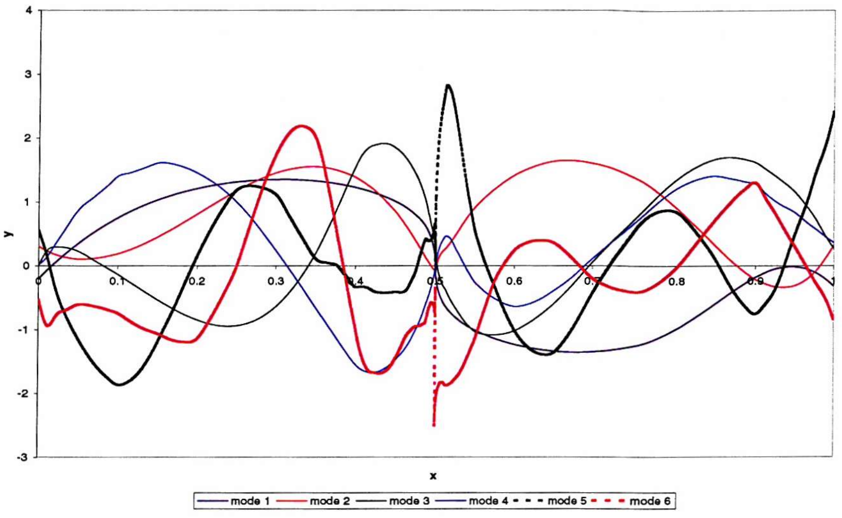

Figure 71: Ortho-Normalised Airfoil Functions Modes 1-6

$$dist = \int_0^1 Abs(f_1(x) - f_2(x))dx \tag{97}$$

where f_1 and f_2 are the y coordinates from the base aero-functions.

The resultant ortho-normalised set of aero-functions, obtained by applying Gram Schmidt normalisation are shown in Figures 71 and 72. Due to the discrete application of the normalisation procedure, a small amount of numerical noise is found to accumulate in each consecutive function. This noise can be attributed to the poor geometrical quality of the original imported sections, and to the round off error inherent by the discrete implementation of the approximation process.

Table 10 shows the total cumulative error in reconstructing all 20 airfoils using n airfoils as basis functions for the ortho-normalisation process. The cumulative error is given by

$$CumulativeError = \sum_{i=1}^{20} dist(upper) + dist(lower) \tag{98}$$

where $dist(upper)$ is the error between the actual upper section and the reconstructed upper section as defined by Equation 97, and $dist(lower)$ is the error for the lower

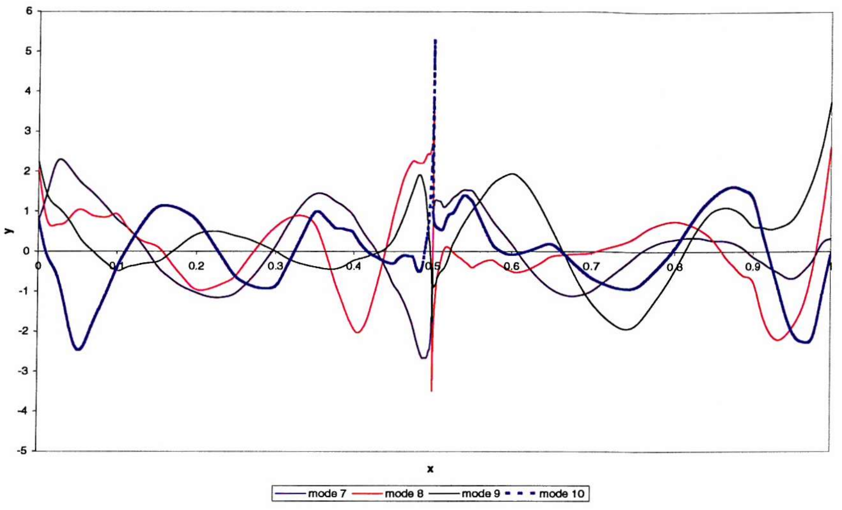

Figure 72: Ortho-Normalised Airfoil Functions Modes 7-10

section as calculated in the same way.

At 20 basis functions, a significant error of 0.0117 is present, and this is largely attributed to numerical noise. However this error threshold is easily reached to within 2% with just eight basis airfoils, and a similar but only slightly larger error is attained with six basis airfoils.

6.4.2 Implementation of Ortho-Normalised Aero-Functions in Shape Optimisation

The first eight orthogonal aero-function modes were used in the shape optimisation problems given in Section 6.1.1. The resultant airfoil for C_L=0.6 test case is shown in Figure 73. The new ortho-normalised functions were significantly easier to implement than the previous ones, and did not need any additional smoothing about the leading edge to obtain this solution. In this implementation, the functions were not able to improve on the result obtained using the previous functions. Once again, large frequencies of non-convergent results were observed during this implementation with

Number of Foils in Base Set	Included Airfoil	Cumulative Error for Reconstructing 20 Airfoils
1	NACA-SC2	1.2913
2	M21	0.4573
3	UI1720	0.3222
4	RAF32	0.1743
5	N6h15	0.1276
6	I7769	0.0985
7	E856	0.0564
8	NACA2301	0.0327
9	RAE5212	0.02711
10	N8h12	0.0246
11	E625	0.0216
12	NCAMBRE	0.0207
13	E793	0.0176
14	E850	0.0123
15	USA408	0.0117
16	N6409	0.0117
17	M5	0.0117
18	E1210	0.0117
19	JN153	0.0117
20	RAF26	0.0117

Table 10: Airfoil Reconstruction Error from using Ortho-Normalised Aerofunctions

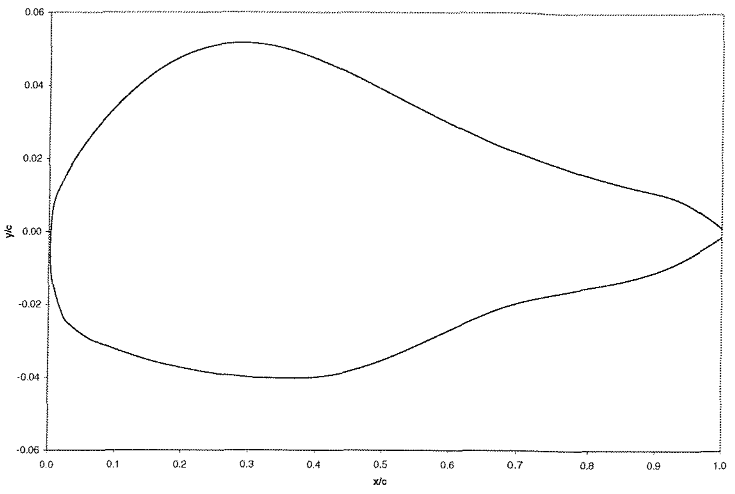

Figure 73: Resultant Section Obtained using Ortho-Normalised Aerofunctions

the GA, and concerns should be noted about the ability of the functions to fully capture smooth leading and trailing edges due to the large amount of numerical noise present in these areas. The severe number of infeasible candidates are the most likely cause for a poor GA solution.

6.5 The Performance of Different Parameterisation Approaches

Several spline based parameterisation techniques were used on a simple airfoil design problem, with the best drag results obtained from each representation, shown in Table 11.

The delta-BSpline approach was the simplest to implement, but was found to be too restrictive for optimal design search. In general the BSpline approaches produced wavy pressure distributions as previously observed by Jameson. By allowing the BSplines poles to move in both x and y directions, much smoother pressure distributions and sections were achieved. The use of NURBS curves with parameter freedom

Representation	Test $C_L = 0.3$	Case $C_L = 0.6$
Delta B-Spline with 50% freedom	0.0059	0.00873
B-Spline with x knot direction fixed	0.0068	0.00881
B-Spline with Moveable (x,y) knots	0.0061	0.00876
Bezier	0.0051	0.00851
Ortho-Normalised NACA Functions	0.0048	0.00846
Ortho-Normalised Aero-Functions	0.0051	0.00845
Extended B-Spline Trial with 12 knots		0.00862
Extended B-Spline Trial with 16 knots		0.00858
Extended B-Spline Trial with 20 knots		0.00861

Table 11: Resultant Best Drag Results Obtained from each Representation

in chordwise, thickness and rational directions have been used by Lépine *et al.* [37], without producing this characteristic, highlighting further search opportunities offered through B-Spline representations. However more than 40 design parameters were required for these implementations.

The leading edge was particularly difficult to define by all the methods used. Some imposition of tangency was required by all representation methods to produce adequate pressure distribution in this area. The Bezier implementation produced the smoothest definition, including some peakedness in the leading edge pressure distribution. Further definition in the region could be achieved either by additional Bezier poles, or through the implementation of a higher fidelity spline method such as NURBS. The orthogonal functions were able to produce a reasonable result in the $C_L = 0.6$ test case, after smoothing was applied to the leading edge, however this solution may reduce its ability to define sharply peaked pressure distributions.

The study of bidirectional airfoils presented the opportunity to examine the suitability of orthogonal functions to a slightly different geometry. The parameterisation was simplified significantly in this problem since only definition for one surface was required, and rotated about 180^o forming the other surface. Suitable results were found for both design points.

The implementation of ortho-normalised aerofunctions demonstrated the ability to adopt orthogonal functions in the description of the entire airfoil without additional smoothing about the leading edge. Despite the advantages offered by this approach, the solution failed to improve on the result found by the ortho-normalised NACA functions. The frequency of infeasible candidates generated by orthogonal representations is the most likely cause for this disruption. Numerical noise generated from both surface inaccuracies in the base airfoils, and round off error in the discrete Gram Schmidt implementation were found to be a significant handicap to the adoption of this approach. Opportunities to further reduce this noise exists through using analytical functions that have similar characteristics to the aerofunctions, and through the reduction of round off error in the ortho-normalisation process.

Overall, the orthogonal functions provided the most dynamic geometries, defining slender sections for the $C_L = 0.3$ design point, and well posed pressure distributions in the $C_L = 0.6$ case. Parameterisation required less than half the number of design variables than the other methods. Doubts however remain on the ability of such functions to define leading edge curvature, and the coarseness of the resultant geometrical definition. The Bezier curve method provided smooth curvature throughout, but with a more limited search range. The resultant geometries agreed with that found using orthogonal functions, although it is expected that the global optima continues to remain elusive, due to the poor evolution convergence observed when applying more high fidelity representations.

6.6 Sensitivity of a B-Spline Representation to Parameterisation

B-Spline and NURBS representations offer the opportunity to adapt shape to a finite degree for any given parameterisation. By increasing the parameterisation through more control points, further shape adaptation can be offered. However, so far B-Spline based implementations have been difficult to implement successfully, in GA search processes.

One idea into why B-Spline representations behave so poorly in GA shape optimisation, is that such representations may lead to a deceptive [15] search landscape.

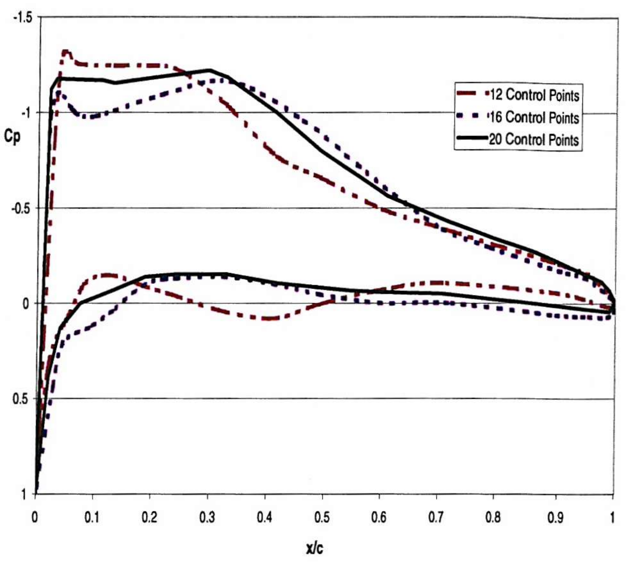

Figure 74: Effect of adding more control points to B-Spline parameterisation

Figure 74, shows the effect of increasing the number of BSpline control points on the final pressure distribution achieved. With just 12 moveable points (ignoring those that are fixed at the trailing and leading edge), a nicely smoothed pressure distribution can be maintained. A total of 20 control points or 40 design variables were required to adequately capture the leading edge pressure peak, showing the enormous expense involved, or using genotype-phenotype mappings that do not allow adaption of the number of genes.

However, the shape obtained using 20 control points was achieved following extensive re-runs of the GA, with each run starting from the previous best solution. In addition, a constraint was also placed on the pressure distribution, limiting the number of curvature inflections to five. In total, 10 re-runs were used to obtain the final result. A more interesting result is that obtained using just 12 control points, six on each surface. With this parameterisation, the solution converged quickly, and a nicely formed pressure distribution was formed as shown in Figure 74.

6.7 Summary

An investigation into the performance of several geometric representation techniques including spline interpolation and analytical functions has been explored in GA based aerodynamic shape optimisation. The popular Bezier spline representation was found to be the easiest technique to implement, and produced smooth sections following optimisation. In contrast, B-Spline curves were found to be the most difficult representation to implement, resulting in unwanted wiggles within the pressure distribution. Following an additional constraint to reduce unwanted pressure distribution fluctuations and additional expensive re-running of the GA from previous solutions, highly satisfactory results were eventually achieved from B-Spline based representations.

The use of ortho-normalised analytical functions with GA's were explored as a means of producing more efficient geometric parameterisations. This type of implementation was able to outperform the performance obtained using spline based representation approaches. The idea was extended to using existing airfoils to construct the orthogonal function set. Although, a successful result was found, concerns were highlighted over the amount of numerical noise generated through the implementation used. Such numerical noise may limit the representations ability to adapt the leading and trailing edges of airfoil sections where the effects of noise are the strongest. Orthogonal functions were also found to generate a high frequency of infeasible designs whose performance could not be determined via the implemented CFD solver. This characteristic is detrimental to the GA evolution process, and is a significant restriction in the ability of these representations to adapt geometry.

Opportunities to increase the ability for spline based representations to adapt, exist through the extension of their parameterisation to cover larger design spaces. However, in the adoption of B-Spline curves, larger parameterisations were found to reduce the ability of the GA to evolve an airfoil shape. A further opportunity for increased adaptivity exists for all representations, through the ability to provide some aerodynamic performance evaluation of candidates that were found to be infeasible in this implementation due to CFD convergence issues. More computationally expensive CFD methods could provide this improvement, but at a considerable computational cost. A potential opportunity for improvement in this field, may exist through the implementation of a genetic repair scheme that is able to mend infeasible candidates to solvable ones.

Chapter 7

Adaption of Chromosome Complexity

The requirement of adaptation considered in this chapter, is the ability to either increase or decrease the flexibility of search offered through the genotype-phenotype map within an adaptive evolutionary process. To incorporate this feature within a GA which generally requires an external mapping between genotype and phenotype, hierarchical structures or components can be used within the definition of the chromosome string.

Hierarchical based chromosome structures have been successfully used in a variety of problems [109]. In previous implementations, strings of child genes are grouped together via a container gene which will be referred to as the parent. By allowing a parent gene to contain other parents, complex hierarchical genetic representations can readily be established. Genetic operators such as crossover and mutation can be applied to these chromosome strings in a similar pattern to that used on simple canonical binary string representations. Mutation operations can alter the value of each gene bit of a child gene, and crossover can allow the exchange of genes between two parents by exchanging similarly ordered genes between parents. These operators can be applied to any level of the hierarchy.

One of the key differences between hierarchical and canonical chromosome representations, is the genetic operation offered by 'Hierarchical Mutation'. Hierarchical mutation can be used to adapt the complexity of the chromosome string by switching on and off child branches or structures within the chromosome definition. Switching off gene branches is a simple operation that requires the affected genes to be removed or disabled from the chromosome structure. When switching on genes, new gene encodings need to be seeded, which is usually achieved through random generation.

Evolution is a gradual process of cumulative small changes, and not one of large random flukes. When using hierarchical encodings in the definition of geometry, switching off strings will generally have a small mutational effect on the overall shape. However, when switching on a gene from its off state, care should be taken to ensure that the resultant phenotype is changed only by a small mutational effect. The hierarchical mapping used by [62], and in [109], showed little concern for the effect of hierarchial mutation on the resultant phenotype, using a random generator to reseed the genes that were switched on. On a geometrical structure in a critical design environment such as aerodynamic shape formation, such random generation can lead to large changes to the resultant phenotype or shape. It would be difficult to see how any chromosome undergoing such transfiguration would survive to the next generation.

Bently [110], addressed such an issue in his work with primitive geometrical objects. To facilitate the addition of new genes and structures into a hierarchial encoding, the new genes were seeded with interpolated values from the existing genes. Since the primitive types used by Bently, exhibited no curvature effects, this method was easy to adopt, and expressed directly in the hierarchical chromosome implementation.

The adoption of an explicit hierarchical chromosome implementation will be difficult for curvature shapes, where more elaborate means of interpolation would be needed to minimise the effect of adding a new gene to the resultant shape. In this work, an object based approach to the same methodology will be implemented for the hierarchical encoding and mutation of advanced geometrical shapes. This approach will allow for the flexible adoption of different geometric representation techniques using the same pattern.

The basis for the framework proposed, is that of a hierarchical relationship between Form, Surface, Curve, and Point. That is to say that a Form is composed of many

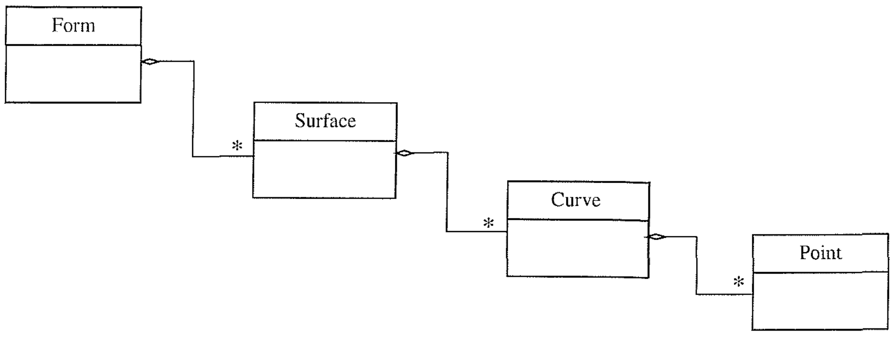

Figure 75: Hierarchical relationship between Form, Surface, Curve and Point entities

Surfaces, a Surface is composed of many Curves, and a Curve is composed of many Points. This relationship is illustrated in Figure 75. By using a hierarchy of entity objects, reproductive operators applied at a high level such as on a curve, are passed down appropriately to lower level entities such as a point. By allowing the genotype objects to contain knowledge of its phenotype mapping, complex chromosome operations such as hierarchical mutation can be facilitated more appropriately while ensuring that a philosophy of evolution through small changes persists. For example, a fifth knot can be added to a B-Spline curve by interpolating its values such that the shape of the resultant curve does not change unless a point mutation is applied as illustrated in Figure 76. A crossover operator can also be added to deal with two parent chromosomes that are hierarchically different to one another. The disadvantage of using an object hierarchy to represent the chromosome structure is the increased amount of programming required to re-implement an encoding for a different representation.

7.1 Hierarchical Chromosome Framework

In order to capture geometric detail within the genotype, it is important to maintain a complete DNA specification of the geometric phenotype, which can be achieved through allowing both mutable and non-mutable values to co-exist within the same encoding structure. Reproductive operators should then copy non-mutable values

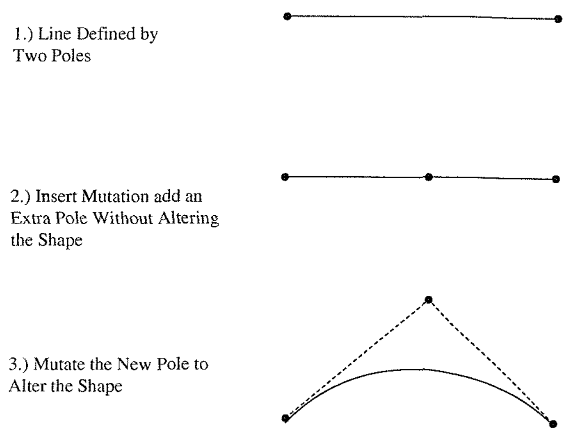
1.) Line Defined by Two Poles
2.) Insert Mutation add an Extra Pole Without Altering the Shape
3.) Mutate the New Pole to Alter the Shape

Figure 76: Adapting a line into a curve using a B-Spline representation, under the Hierarchical Framework this adaption can use two mutations, one to add the extra genes (new Point), the other to mutate the new genes

into the offspring chromosome structure directly without modification, and treat mutable values through the appropriate reproductive operators such as crossover and bit mutation.

A sample chromosome structure is given in Figure 77 for an object based hierarchy capable of describing the evolution given in Figure 76. Each parent entity is responsible for activating reproductive operators on its child objects, including seeding appropriate values when hierarchical mutation is called in order to reduce the effect of hierarchical mutation to small changes only. An example of how hierarchical mutation will cause the structure of the chromosome to change is given in Figure 78, and the final curve is described following a second mutation in Figure 79.

To implement the new hierarchical chromosome framework within a GA, the GA used in Chapter 4 was re-implemented using the JAVA programming language. A basic description of the new implementation is described in Appendix D, which illustrates the important relationship between the evolution process encapsulated by the GA, and the Hierarchical DNA interface implementation described here. The migration from C++ to Java was straightforward, assisted through the exploitation of the vast and well documented Java 1.3 API [111]. The Java API provides significantly more

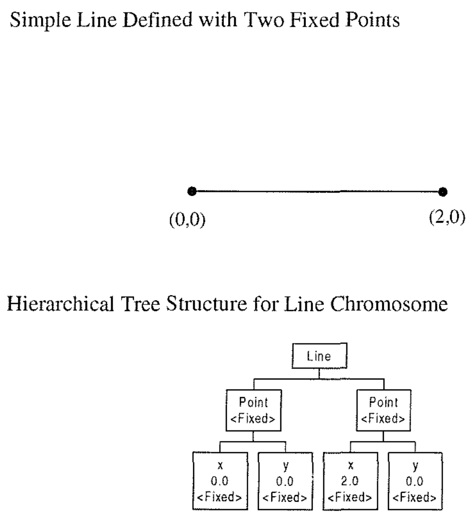

Figure 77: Hierarchical Implementation of a Simple Curve Chromosome

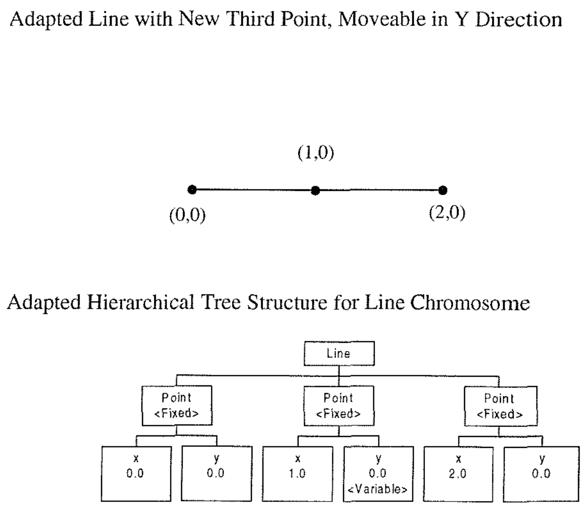
Adapted Line with New Third Point, Moveable in Y Direction
(1,0)
(0,0)
(2,0)
Adapted Hierarchical Tree Structure for Line Chromosome
Line
Point <Fixed>
Point <Fixed>
Point <Fixed>
x 0.0
y 0.0
x 1.0
y 0.0 <Variable>
x 2.0
y 0.0

Figure 78: Adaption of Hierarchical Curve Chromosome by Adding New Genes in the Form of an Extra Point

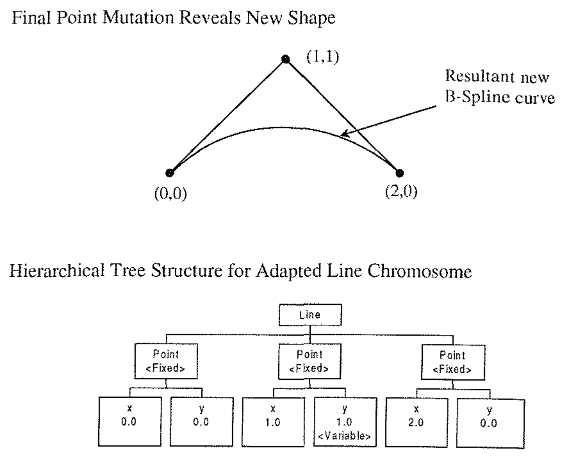
Final Point Mutation Reveals New Shape
(1,1)
Resultant new B-Spline curve
(0,0)
(2,0)
Hierarchical Tree Structure for Adapted Line Chromosome
Line
Point <Fixed>
Point <Fixed>
Point <Fixed>
x 0.0
y 0.0
x 1.0
y 1.0 <Variable>
x 2.0
y 0.0

Figure 79: New shapes can easily be created once new points have been added

support for container types such as vectors, strings and hash maps than STL, allowing development to focus clearly on the implementation logic of the GA rather than on extra support for value types.

The key reason for choosing the Java platform over C++ for the implementation of the hierarchical GA, is Java's ability for Serialisation and Reflection, used in Java Remote Method Invocation (RMI) for distributed computing over networks. Unlike other distributed technologies such as Corba 3.0 [112], Java is able to completely serialise and marshal its objects at runtime so they can easily be communicated between distributed computers, requiring only a small amount of extra programming. Because the new chromosome structures contain a collection of problem specific objects to describe the genotype, other distributed technologies such as Corba or MPI, would require an external description of the signature types that are to be communicated between processors, burdening distributed development.

7.1.1 A Persistent Asynchronous Distributed Computing Environment via JINI-JavaSpaces

To continue to evaluate expensive objective functions over distributed computing networks, Java's object serialisation capability enables the definition of phenotype orientated chromosome objects without the need to define further socket serialisation mechanisms. Distributed communication amongst Java processes is generally made available through socket libraries, or RMI. In addition to these basic services, one other high level service API has been made available for network based communication. JINI technology is orientated around distributed services and provides support for network services such as lookup, communication, synchronisation, leasing, distributed events and notification, and distributed transactions. One powerful service provided through JINI, is JavaSpaces.

The JavaSpace model provides persistent object exchange areas in which remote Java processes can coordinate their actions and exchange data. Motivated by the Linda language [113], JavaSpace views a distributed application as a collection of processes cooperating through the flow of objects into and out of one or more distributed spaces. A space is a shared, persistent network repository for Java objects. Instead of the process talking to other processes directly, they exchange data by reading, writing and taking objects from a space. In support to the JavaSpaces concept, JINI provides the additional services such as lookup, events, leasing, transactions, etc.

A space based object is defined from the Entry interface. Instance values can be used to look up entries within a space. Hence an Entry instance with a parameter 'String status', will allow the lookup of objects via string matches such as $status =' HelloWorld'$. In fact, this lookup mechanism is extended to all Java value types including objects. It is this envelope or 'template' that provides the important capability to read, write, take and receive notifications on specific object events.

JavaSpaces provide a basic scalable and persistent service on which to base a Java implementation of the farm-worker model described in Chapter 5.2.2. A basic spaced based model can be created as described in Figure 80.

```
1. If GA
    (a) Look Up 'work' JavaSpace.
    (b) Add objects that require evaluating to space with status='Evaluate'.
    (c) Remove all objects from space with status='Evaluated'.
2. If Worker
    (a) Look Up 'work' JavaSpace.
    (b) Register with space for object additions where status='Evaluate'.
    (c) Take object.
    (d) Evaluate.
    (e) Return object to space with status='Evaluated'.
```

Figure 80: Simple JavaSpace Model of Farmer-Worker

7.1.2 A Robust Farm-Worker Spaced Based Implementation

The above algorithm will enable the Farm-Worker features needed for the distributed GA. In order to provide a more robust asynchronous evaluation system, the following requirements have been included in the following implementation.

- Persistent availability of a 'work' object for a given length of time or until it has been evaluated, regardless of network and computer failure.
- Object evaluation can be passed to selected resources.

These requirements basically ensure that regardless of computer, application crashes or network failure, for as long as the given space is available, once a 'work' object enters the space, it is available for evaluation until evaluated, is taken for evaluation, or expired. Even if the job submission process has failed, the object remains available. If a long enough lease time is allocated to the work object, then evaluation must be allowed to complete. This applies even if a remote process has removed the object for evaluation, but failed to return the completed result due to some failure. The second requirement allows the job submitter to define the type of resource that should carry out the work.

For the GA, the basic algorithm for submitting job objects remains relatively unchanged to that in Figure 80. Only the additional specification of a maximum work lease time is required, as well as a destination indicator that describes what type of worker should carry out the work. On the lease expiring, the object will be removed from the JavaSpace.

For the worker, the problem of object persistence within the JavaSpace in the event of partial failure is handled through the JINI transaction and lease model. The worker process removes jobs from the space under a transaction, barring other processes from accessing or reading the object. Upon completion of the objective function evaluation, the fitness and constraint violations are recorded within the object envelope, and the status value set to 'Evaluated'. The object can then be written back to the space under the same transaction. Applying the transaction 'commit' operation, applies the transacted JavaSpace event to the space. The object is now available to other space clients to read and remove. In the event of partial failure such as process crashes or JavaSpace/GA or Worker computers being switched off in an uncontrolled environment, the original object will be fully restored when the transaction lease eventually expires. This model implies that if an object is given unlimited lease lifetime, and that a worker process is available to carry out objective function evaluation, a fully completed evaluation will always be carried out regardless of the number of partial failures.

The final Farm-Worker algorithms are available in Figures 81 and 82.

7.2 Hierarchical Crossover and Adaptive Mutation of a Line

A curve described by a collection of straight lines between points, provides a simple but practical problem on which to examine the genetic interaction between adapting the complexity of the genotype on its phenotype within an evolutionary process. The Hierarchical implementation is shown in Figure 83. The Point objects contain the underlying DNA sequence if the chromosome, and the objective of the curve object, is to facilitate calls passed from the GA, onto these underlying points. Two of the most important methods that Line must implement are, Crossover, and Mutation, which allow the structure of the chromosome to adapt by adding or deleting genes, as well as to alter the genes already present.

```
1. Look Up 'work' JavaSpace.
2. Create CounterEntry instance.
3. Set cursor=0, process=myid.
4. Write CounterEntry to space.
5. ForEach Genome in population
    (a) Create GenomeEntry instance.
    (b) Set status='Evaluate'.
    (c) Set type='xfoil'
    (d) Set lease='unlimited'.
    (e) Set process=myid and id=GenomeId.
    (f) Create Transaction with short lease.
    (g) Write entry to space under transaction.
    (h) Take CounterEntry from space under transaction.
    (i) Set $cursor+ = 1$ and write CounterEntry to space under transaction.
    (j) Commit transaction.
6. While $CounterEntry > 0$
    (a) Request events for GenomeEntry write where process=myid and status='Evaluated'.
    (b) Upon event notification
        i. Create transaction with short lease.
        ii. Take GenomeEntry from space where process=myid and status='Evaluated' under transaction.
        iii. Take CounterEntry from space under transaction.
        iv. Set $cursor- = 1$ and write CounterEntry to space under transaction.
        v. Commit transaction.
7. Return.
```

Figure 81: JavaSpace Farm Implementation within GA Evaluate

7.2.1 Hierarchical Crossover

Hierarchical Crossover mimics the crossover operation usually performed on the simple binary string chromosome described in Chapter 4.1.4, but within the hierarchical tree structure. Hierarchical Crossover consists of two genetic exchange operations between parents. Branch exchange illustrated in Figure 84, essentially cuts the gene sequence between two different parent genes. In this operation, all parent genes to the right of the cut within the same branch are swapped with the corresponding genes of the second Genome. All children branches of affected parent genes are also transferred to the second Genome. The second crossover operation available, allows a gene of any branch level to be cut as illustrated in Figure 85. A portion of this gene to the right of the cut, along with all remaining genes to the right of the cut are transferred to the second Genome.

```
1. Look Up 'work' JavaSpace.
2. While True
    (a) Request events for GenomeEntry write where type='xfoil' and status='Evaluate'.
    (b) Upon event notification
        i. Create transaction with long lease.
        ii. Take GenomeEntry under transaction.
        iii. Create WorkerProcess entry.
        iv. Set worker=myid and id=GenomeEntry.id and write entry (no transaction).
        v. Evaluate GenomeEntry and write fitness and constraint violations to GenomeEntry.
        vi. Write GenomeEntry to space under transaction.
        vii. Commit transaction.
        viii. Take WorkerProcess (no transaction).
3. Return.
```

Figure 82: Algorithm for JavaSpace Worker

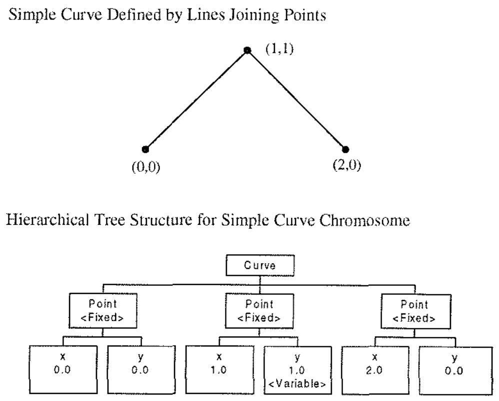

Figure 83: Chromosome Structure for a Set of Connected Lines

In a simple implementation of this process, the position of the crossover point is determined by only one parent. If the crossover point is not found to exist on a mutable or switched on gene on the other parent, then the cross-point will mark a position from which all genes are swapped without crossover being performed on a gene. If the gene belonging to the second parent is also mutable, then the task is passed down to their implementations. N-Point crossover is achieved by calling the crossover function N times.

The algorithm for Hierarchical Crossover of a Line is summarised in Figure 86. The Point crossover method is summarised in Figure 87.

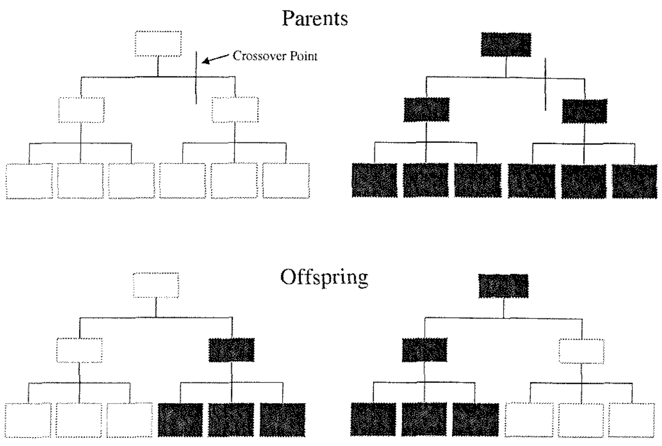

Figure 84: Parent gene exchange via sub-tree swapping

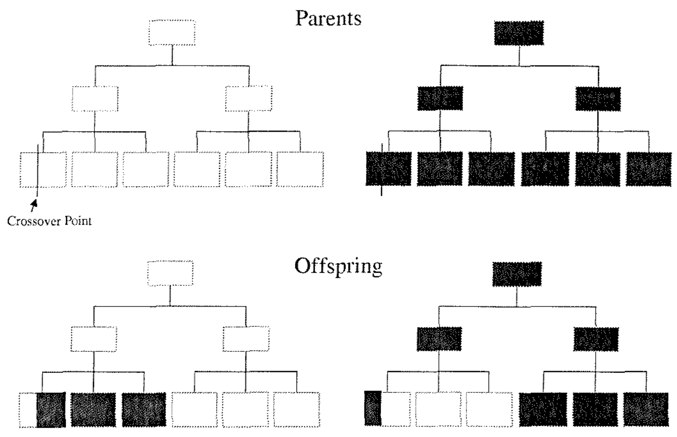

Figure 85: Hierarchical crossover at sub-level

```
1. create array of mutable points
2. select random point from mutable point array
3. if ( chosen point is available in other parent ) then
    (a) crossover genes between points
    (b) swapover remaining points in curve
4. else
    (a) swapover points in curve from chosen point
```

Figure 86: Hierarchical Crossover Algorithm

```
1. randomly pick either x or y as the mutable value if both their mutable methods return true, otherwise pick the mutable value
2. select same value on other parent
3. pass one value to others crossover method
4. if both x and y are mutable, the non-crossed value can be swapped on a random basis
```

Figure 87: Point Crossover Implementation

Improved Hierarchical Crossover

To increase schema sampling through a more productive crossover operator, a filter can be applied such that only common genes that are present and available between both genomes are passed to the gene crossover operation. To achieve this, the chromosomes are searched to create a list of common indices. Each index represents a mutable gene that is present in both genome encodings. The index that is to be used to pass the respective points to the genes crossover operation, is then randomly chosen from this List. The Algorithm used to filter the genes, and its use in the modified Hierarchical Crossover operation is shown in Figure 88.

This algorithm is now almost identical in effect to Benley's [110], Hierarchical Crossover implementation. In Bentley's algorithm, a more complex search was carried out to find a point of similarity between the two parent chromosome strings. This search was carried out by randomly choosing a gene, and attempting to discover the same gene on the second parent. If the gene was not found, a search was conducted to find all similar genes within the same hierarchy. A random gene would then be sampled from this list. If no similar genes were found, the search would move up in the hierarchy until the top level is reached without any crossover occurring within a

```
1. for each point
    (a) if point exists in other parent curve
        i. add point to common list
2. if Size( common point list ) ¿ 0 then
    (a) select random point from common point list
    (b) crossover genes between points
3. else
    (a) select random sub-tree from first chromosome
4. swapover remaining points in curve
```

Figure 88: Algorithm for Hierarchical Crossover about Common Gene Points

gene. The algorithm implemented here facilitates a more effective crossover operation, as it will always provide a set of common genes to the gene crossover operator, if such genes exist within the curve structure. If no common genes are found, then a simple gene swap is performed between chromosomes from a randomly chosen point.

7.2.2 Hierarchical Mutation - Gene Addition and Deletion

The mutation operators made available to assist the evolution of both chromosome structure and genes are:

1. Gene bit mutation
2. Sub-branch inversion
3. Child node deletion
4. Child node addition

In some hierarchical chromosome implementations, bit mutation, node deletion, and node addition are implemented equally as part of the same operation. This usually entails that the gene switches which determine whether a gene is active or not, become part of the chromosome string. These additional gene components, undergo all breeding operations during reproduction. This introduces further chromosome complexity, increasing the overall length of the chromosome string. In this implementation, the determination of whether a gene is active or not, remains hidden from the main breeding processes. Valuable genetic processes such as crossover and inversion do not wasted genetic operations altering the contents of genes that are not active, leaving this process solely to a separate mutation operation, 'Sub-Tree Mutation'.

Gene Bit Mutation and Inversion within an Object Hierarchy

Gene bit mutation simply tries each mutable bit in the chromosome string recursively for mutation. For each trial, a random number $rand$ is drawn in the range $[0, 1)$. If the condition, $rand < probabilityMutation$ is satisfied, then that bit is mutated, and the next one sampled. The hierarchical tree is traversed from parent to child, with all genes within the chromosome string sampled.

In inversion, a condition $rand < probabilityInversion$ is tested for each chromosome. If the condition is satisfied, each child branch belonging to the chromosome head parent is chosen for inversion. The node inversion operator is called passing the second Genome's chromosome as an argument. Each node's inversion operator repeats this process, recursively applying the operator on all child nodes. Under a node branch, an invertible bit within randomly selected node is selected to become the starting point for the inversion process, and the end point chosen from within any node under the same parent branch. All bits between the start and end point are inverted as described in Chapter 4.1.4. In this implementation, the end point can be placed to either the left or right hand side of the start point.

Sub-Tree Mutation Operator

For a simple line described through a composition of points, the process for point addition and deletion are relatively simple. For sub-tree mutation involving the deletion of a point object, the point is simply removed (the child node is switched off), and the resultant curve is altered by only a small localised amount as shown in Figure 89. For insertion, the gene is switched on, and the seeded position of the point is calculated through the algorithm briefly described in Figure 7.2.2.

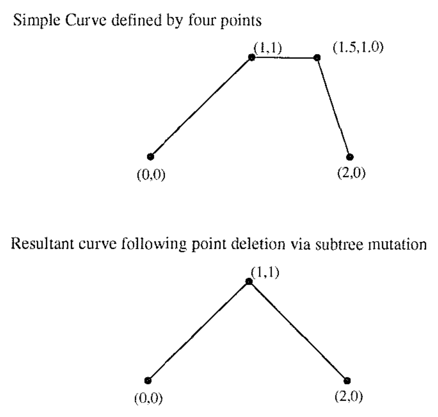
Simple Curve defined by four points
(1,1)
(1.5,1.0)
(0,0)
(2,0)
Resultant curve following point deletion via subtree mutation
(1,1)
(0,0)
(2,0)

Figure 89: The effect of sub-tree deletion on a simple curve

Figure 90: Line Curve Point Insertion Algorithm

1. Find a, b such that $y = ax + b$ for all points between $P_1 and P_2$.
2. Find range of x $[x_{min}, x_{max})$ that satisfies y $[y_{min}, y_{max})$ where $y = ax + b$.
3. Randomly choose a value x within the valid range, and yield y.
4. Seed newly enabled gene with (x, y) values

The effect on the curve, and chromosome structure following point insertion on the curve shown in Figure 83, is illustrated in Figure 91.

7.2.3 Application of GA to evolve complexity of a simple curve

A simple line reconstruction problem has been used to explore the capability of the hierarchical scheme to evolve shape complexity. Three problems are considered involving the reconstruction of lines defined by the coordinates given in Table 12. The objective of the problem is to re-construct each solution using the hierarchical GA described above, from a two point starting solution with coordinates $(0, 0)$ and $(1, 0)$. The encoding used to define the scope of the chromosome map is shown in Table 13.

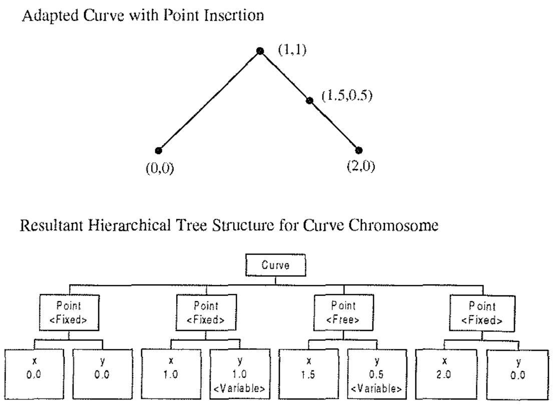

Figure 91: The effect of sub-tree addition on the curve and defining chromosome structure

Point 1	Point 2	Point 3	Point 4	Point 5	Point 6
(0,0)	(0.5,1)	(1,0)			
(0,0)	(0.25,0.5)	(0.5,1)	(1,0)		
(0,0)	(0.25,0.5)	(0.5,1)	(0.75,0.5)	(0.85,-1)	(1,0)

Table 12: Coordinates of the line problems considered

Distance Between Two Curves

The objective fitness function is constructed from the distance between two curves. This distance is taken as the actual cartesian area between them as given in Equation 99

$$error = \int_0^u \sqrt{(P_{1_x}(x_s) - P_{2_x}(x_s))^2 + \left(P_{1_y}(x_s) - P_{2_y}(x_s)\right)^2} \delta x_s \qquad (99)$$

where u is the normalised parametric length of the curve (usually normalised to 1.0), and $P_{x,y}(x_s)$ is the point given at the x-coordinate found at parametric length s on the first curve. Equation 99, applies only for curves that cover the same range in the x-axis. For the implementation used in this chapter, the mapping of $P(x,y) \rightarrow u$ is approximated to $f(x,y)$, and the error is discretely approximated by Equation 100.

Point	Line 1	Line 2	Line 3
1:min 1:max	Fixed (0,0)	Fixed (0,0)	Fixed (0,0)
2:min 2:max	(0,0) (1,1)	(0,-1) (0.35,1)	(0,-1) (0.35,1)
3:min 3:max	Fixed (1,0)	(0.35,-1) (0.65,1)	(0.35,-1) (0.6,1)
4:min 4:max		Fixed (1,0)	(0.6,-1) (0.8,1)
5:min 5:max			(0.8,-1) (1.0,1
6:min 6:max			Fixed (1,0)

Table 13: Hierarchical Encoding Limits for Curve Points

$$error = \sum_{n=1}^{n=maxits} \sqrt{\left(f_{1_x}(x_{s_n}) - f_{2_x}(x_{s_n})\right)^2 + \left(f_{1_y}(x_{s_n}) - f_{2_y}(x_{s_n})\right)^2} \cdot dx \qquad (100)$$

where $dx = \frac{1}{2}(x_{s_n} + x_{s_{n-1}})$ and $s_n = n \cdot \frac{u}{maxIterations-1}$.

The final objective function was constructed as $1 - error$ where $error$ is found using equation 100.

Analysis of Line Adaptation using a Hierarchical Encoding

The performance gain due to hierarchical adaptivity, has been measured against a fixed canonical binary string encoding for the three given test cases. In the case of hierarchical tree adaptivity, an additional tree mutation rate was introduced with mutation rates of 5%, 10% and 15% tested. The performance improvement gained through the use of adaptive chromosome encoding for the three test cases are shown in Table 14, where performance improvement is measured against a traditional fixed binary string encoding of Table 13.

For case 1, a line defined by just three points, the performance measured indicates a substantial performance loss when applying Hierarchical Mutation. Similar results can be seen in the 2nd test case involving 4 points. In the more complex six point

Test Case	No Mutation	5% Mutation	10% Mutation	15% Mutation
Case 1	100%	61%	64%	62%
Case 2	100%	69%	72%	68%
Case 3	100%	104%	113%	102%

Table 14: Performance of Adaptive Line Tree Mutation when Compared with a Simple GA Implementation

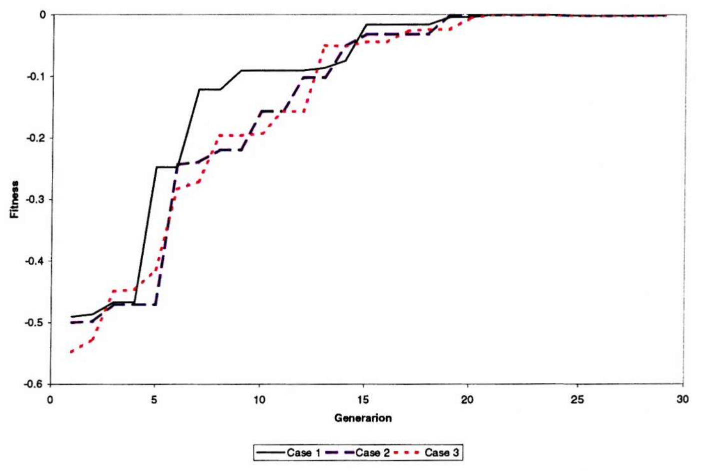

Figure 92: Evolution of fitness

problem, a small performance increase was seen. The fitness evolution for each test case with tree Mutations 10% are shown in Figure 92. In all three test cases, the adaptive encoding required approximately 15 generations to converge.

To provide some analysis into the effect that the adaptive encoding has on the evolution convergence, the number of unique individuals per generation is shown in Figure 93, with tree Mutation = 10%. In the first 10 generations there are significantly fewer unique individuals when tree adaptation is applied in comparison to the fixed encoding. GAs are believed to require a high rate of schema sampling, and a reduction in unique individuals would reduce evolution performance. For the more complex Case 3 problem, the standard GA was observed to converge quickly, with the number of unique individuals available falling sharply before the optimum is found. For the adaptive GA, the large rise in available unique individuals may assist the evolution process to converge on the optimum.

Figure 94, shows the number of common genes shared between mating parents. In

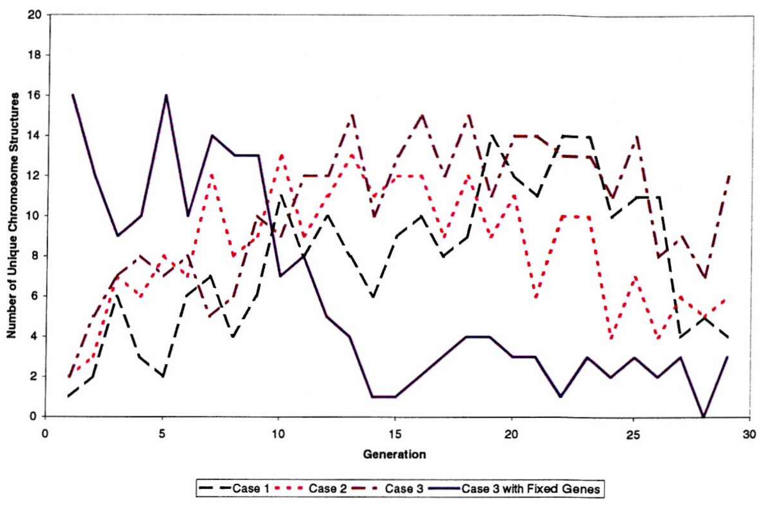

Figure 93: Number of unique individuals in each population

Figure 95, a measure of crossover efficiency is shown, given by the percentage of new children that differ by at least one allele bit from either parent. For the standard fixed GA, crossover is most productive in the first 10 generations while for the adaptive encoding, crossover is most efficient after 10 generations when the number of common genes between breeding parents increases.

7.3 Adapting the Complexity of a B-Spline curve

To extend the notion of chromosome complexity adaption demonstrated in Section 7.2.3 to the difficult problem of airfoil design, a B-Spline representation is adopted to facilitate adaption of the genetic language. The success behind using the B-Spline curve and surface representation to achieve the same aim depends on the fact that there are an infinite number of defining B-Spline polygons that can describe exactly the same shape.

With B-Splines, the flexibility of the defining curve can be adjusted either by raising or lowering the degree of the curve, or by inserting / removing additional knot values into the defining knot vector. This second approach is considered here as it allows the gradual adaption of shape flexibility.

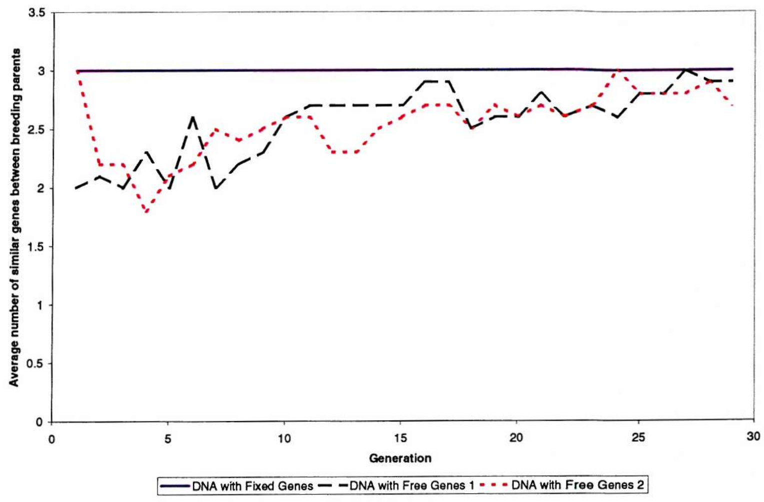

Figure 94: Number of similar genes shared between breeding parents

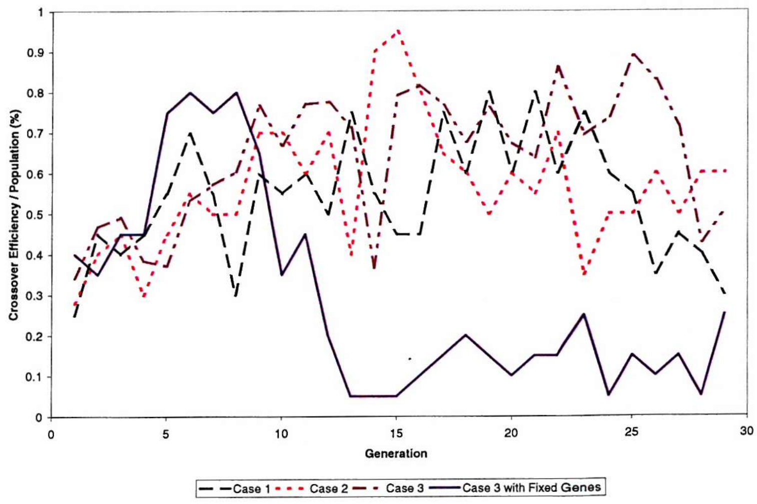

Figure 95: Crossover efficiency

7.3.1 B-Spline Knot Insertion and Deletion

An essential property of this work, is a respect for gradual evolution. It is important to ensure that the adaptation of chromosome complexity only affects the phenotype by a small amount. This can be achieved with B-Splines through a process of knot insertion. The subsequent effect of knot insertion, is to split the piecewise polynomial segment for a given knot interval into two segments. Several methods are available for knot insertion, one of the most popular is the Oslo algorithm, implemented by Cohen et al. [114], which allows for the simultaneous insertion of multiple knot values into the defining knot vector. The insertion of just a single knot, will lead to a slight local adjustment of neighbouring control points. In the genetic algorithm, such movement may be constrained due to the parameterisation of the chromosome genes which require upper and lower bounds. It may therefore become necessary, to recompute such knot insertion for different knot values. Therefore a simpler knot insertion algorithm has been implemented here, based on an algorithm first proposed by Böhm [115], that allows the insertion of a single knot value n times into the knot vector.

For an original curve $P(t)$ defined by

$$P(t) = \sum_{i=1}^{n+1} B_i N_{i,k}(t) \tag{101}$$

with a knot vector

$$[X] = [x_1, x_2, \cdots, x_{n+k+1}] \tag{102}$$

insert a knot into the interval $[x_j, x_{j+1}]$. The objective is to determine the new defining polygon vertices C_j such that $P(t) = R(t)$ and

$$R(t) = \sum_{j=1}^{m+1} C_j \bar{N}_{j,k}(t) \tag{103}$$

where $m = n + 1$ and the new knot vector with the additional knot y_j is

$$[Y] = [y_1, y_2, \cdots, y_j, \cdots, y_{m+k+1}] \tag{104}$$

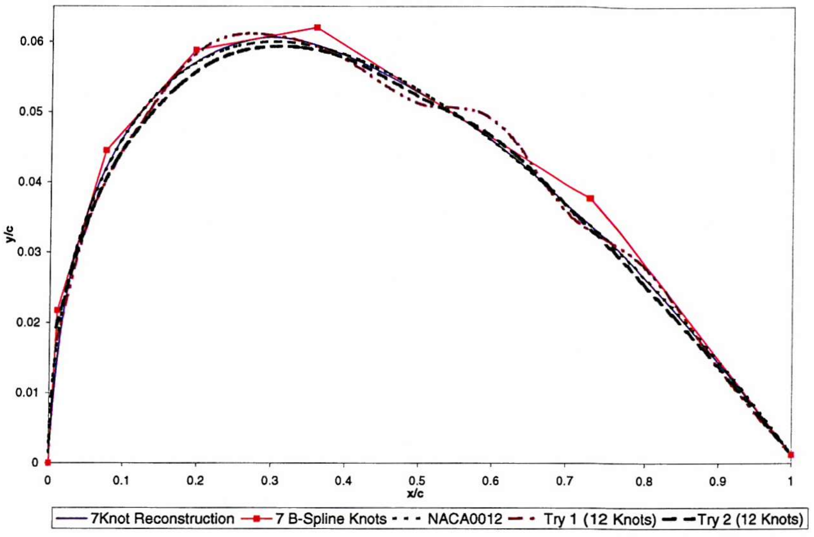

Figure 96: Reconstruction of NACA0012 using GA with Fixed Encoding

From Böhm's knot insertion algorithm, the new polygon vertices are

$$C_j = \sum_{i=1}^{n+1} \alpha_i B_i, 1 \le i \le n \tag{105}$$

where α_i's are given by

$$\alpha_i = \begin{cases} 1 & i \le j-k \\ \frac{y_j - x_i}{x_{i+k} - x_i} & j-k+1 \le i \le j \\ 0 & i \ge j+i \end{cases} \tag{106}$$

7.3.2 Implementation of Knot Insertion Within Complexity GA

To examine the effectiveness of implementing the adaptive B Spline representation, a geometry reconstruction problem is examined. The reconstruction of the upper surface of a NACA0012 airfoil is considered using 7 and 12 B-Spline knots. For the adaptive case, all five knots are switched off initially requiring activating and seeding on tree mutation which is set with a probability of 10%.

Figures 96 and 97 show the resultant sections and B-Spline polygons obtained following 100 generations evolving a population of 200 members. For the fixed encoding

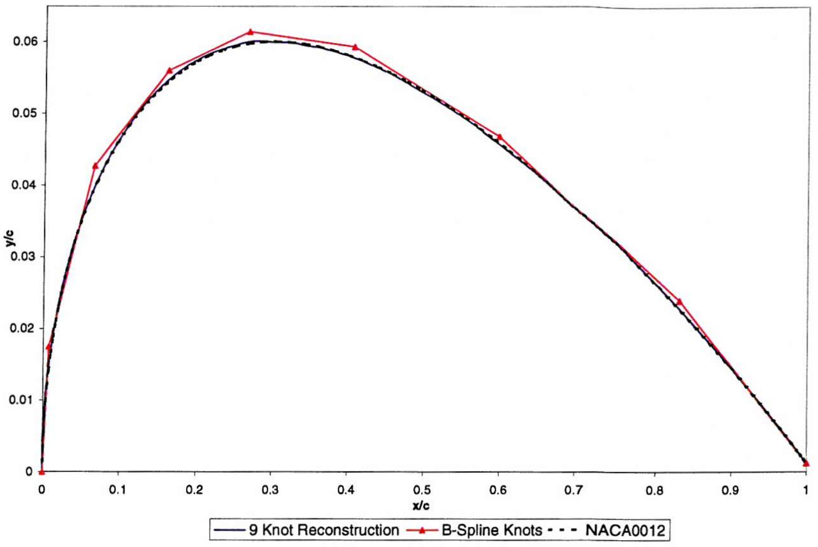

Figure 97: Reconstruction of NACA0012 using GA with Adaptive Encoding

attempts shown in Figure 96, the GA failed to reconstruct the curve to an adequate quality, while the adaptive encoding provided a more satisfactory result, requiring 9 knots from 12 available.

One interesting aspect of the investigation, is the inability of the fixed 10 knot solution to capture the geometric detail of the curve. From the objective function convergence shown in Figure 98, the fixed 12 knot solution was found to be inferior to that of the 7 point solution. This was a surprising result since B-Splines are commonly used for curve reconstructing in CAD using Least Squares Minimisation techniques. To verify the feasibility of the solution, a 12 knot B-Spline was successfully used to reconstruct the given curve using Powell's Direct Search [19], starting from an approximation curve using NACA0012 curve ordinates as knot vectors. Powell's search was used to find the set of knot positions that minimises reconstruction error as defined in Equation 100. The reconstruction result is shown in Figure 99. Powell's method was able to reconstruct the airfoil section to a more satisfactory result than that found by the Genetic Algorithm. The poor Genetic Algorithm performance found in comparison to that of the Powell method, may indicate a large element of GA deception [15] that is introduced by the B-Spline representation. From the Schema Theory, GA deception denotes the inability of a Genetic Algorithm to traverse the objective landscape due to a lack of correlation between improvements in fitness within the search landscape,

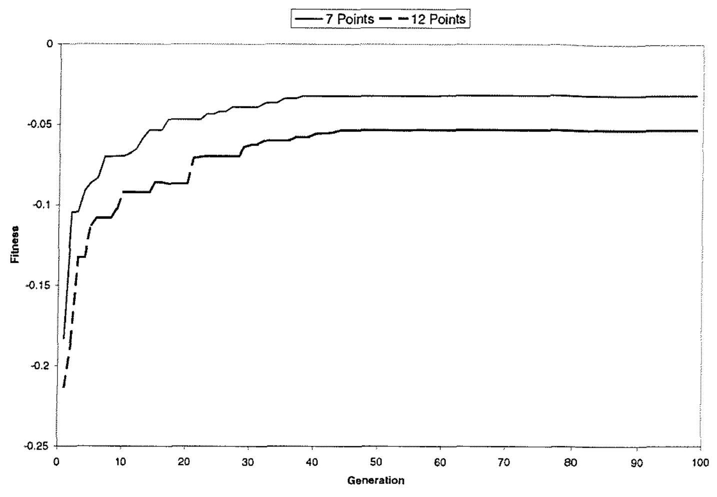

Figure 98: NACA0012 Reconstruction Convergence for Fixed Gene Encoding

and the Schema hyperplanes that represent good chromosome building blocks. Since Genetic Algorithms search primarily using chromosome crossover, if an improving fitness landscape cannot be found through the sampling and exchange of good schema(hyperplanes in the landscape) using this mechanism, then the landscape is denoted as 'GA Deceptive'.

7.3.3 Application of Adaptive Encoding on Aerodynamic Shape Optimisation

An adaptive B-Spline encoding has been applied to the aerodynamic design problem studied in Chapter 6.1.1. The upper and lower surfaces were represented by separate spline curves. The hierarchical chromosome encoding has been parameterised to allow 10 mutable knots to exist on each curve, giving a total of 40 design points available for optimisation (each knot defined by an x and y coordinate). The initial starting point used for the GA consisted of only three moveable points on each curve.

The GA was configured in the same manner as in Section 6.1.2, with the additional tree mutation rate set at 10%. The results from five GA runs for each of the two design conditions are shown in Table 15. Seven out of the ten results shown were able to exceed the performance of the final designs obtained with Bezier and Orthogonal

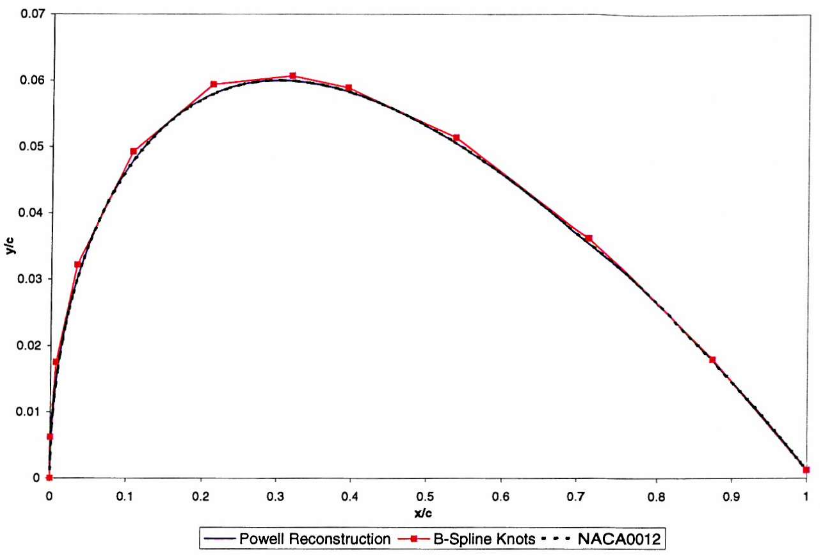

Figure 99: Reconstruction of NACA0012 using Powell's Direction Set Method

Test	$C_L = 0.3$	$C_L = 0.6$
1	0.0047	0.00782
2	0.0045	0.0078
3	0.0050	0.0083
4	0.0051	0.0079
5	0.0048	0.00792

Table 15: Resultant Airfoil Performance Obtained Using Complexity BSpline Encoding

Aerofunction representations. Previous results obtained in Section 6.1.2 were $C_D = 0.0048$ for the $C_L = 0.3$ case, and $C_D = 0.00845$ for the $C_L = 0.6$ case.

The convergence plot of the best run for the $C_L = 0.6$ test is shown in Figure 100. The average number of genes per generation is shown in Figure 101, alongside the number of genes used to define the elitist member of each generation. One interesting aspect shown by this result, is the limited number of B-Spline knots selected by the GA to define the optimum section. The initial investigations in the the role of B-Spline representations in shape optimisation given in Section 6.2.1, had always assumed that larger design spaces would yield more highly adapted results. The final resultant section given in Figure 102, shows a more simply defined BSpline polygon choosing to use only 9 knots for the upper surface. The resultant pressure distribution obtained from the optimised section is shown in Figure 103. The pressure distribution does not

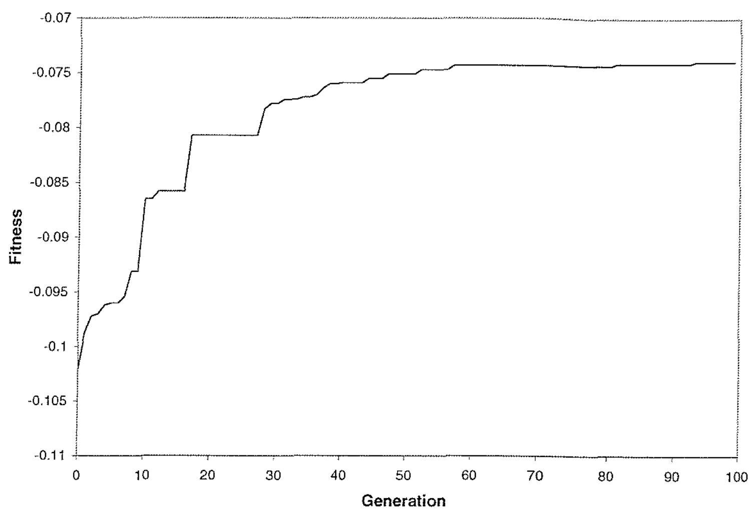

Figure 100: Airfoil Optimisation Convergence Using Complexity BSpline

exhibit the wavy characteristics found in Figure 49. In addition, a different pressure distribution has been obtained with the loading focused around the leading edge. An improved leading edge representation is one factor that would allow such a loading to exist and may be a factor in this result.

7.4 Summary

The concept of complexity adaptation has been explored in this chapter as a potential means of improving the ability to form highly adapted shapes using a GA. The idea of complexity adaption was implemented through a hierarchical component based chromosome encoding, that allowed one component to remove or add another dependent component. One significant divergence that was made to previous implementations of this concept, was the requirement on the chromosome operators to ensure that evolution is maintained through the accumulation of small changes. This required that new component structures, formed through complexity mutation are initially seeded with values interpolated from the phenotype.

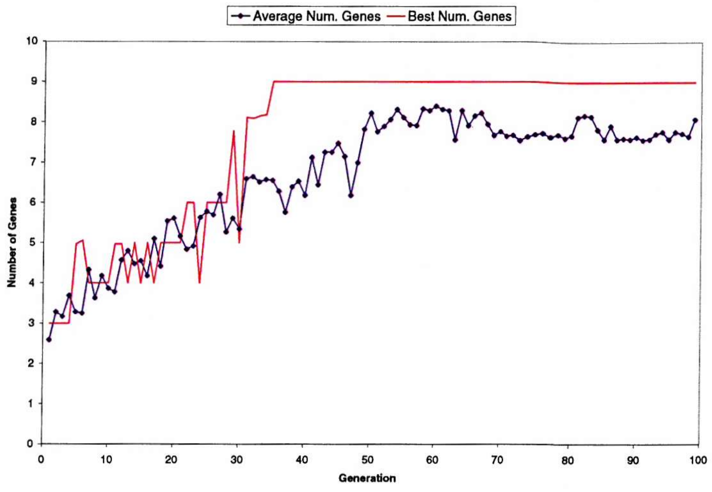

Figure 101: Number of Genes Used per Generation in the Evolution of $C_L = 0.6$ Complexity Airfoil

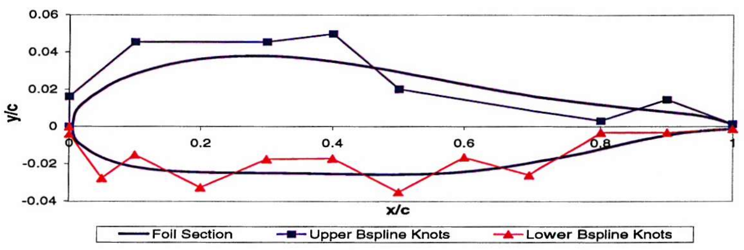

Figure 102: Resultant Airfoil Section from Using Complexity BSpline Representation

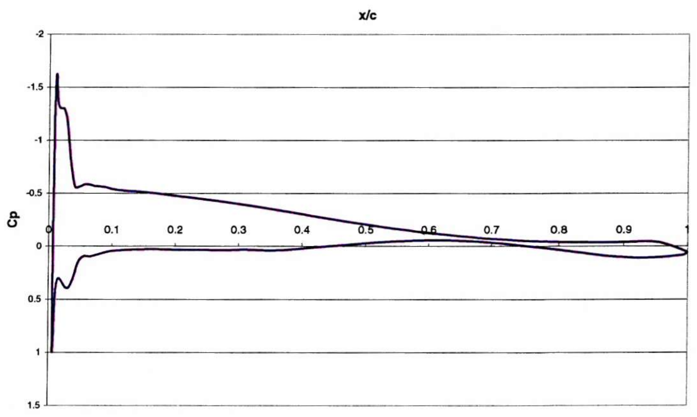

Figure 103: Resultant Airfoil Pressure Distribution Obtained Using Complexity BSpline Representation

The implementation of the new hierarchical encoding, required significantly more effort than the traditional binary string chromosome. On application to simple geometry reconstruction problems, the new encoding was found to require several generations before the evolutionary breeding operator 'crossover' became effective. Although this feature acted as a handicap on simple parameterisations, on more complex problems, the hierarchial encoding was found to outperform traditional fixed chromosome strings.

On application to aerodynamic shape optimisation using B-Spline representations, a knot insertion algorithm was implemented to ensure integrity in the phenotype during complexity adaptation. The implementation was found to successively outperform previous representation trials, while selecting to use fewer design points in the encoding.

Chapter 8

DNA Repair for Infeasible Offspring

The high frequency of bad solutions that provide no useful information to the optimisation cycle can lead to weak GA convergence. In aerodynamic shape optimisation involving sensitive CFD analysis, this problem can become acute where the CFD tool is unable to analyse design candidates due to numerical convergence problems.

In nature, genetic repair is used to mend broken combinations of DNA formed during reproduction. Implementing a repair scheme within a GA could significantly improve the rate of sampling schema within the candidate solution population, and hence the adaptiveness and search efficiency of the GA.

Genetic repair in GAs is difficult to implement, as it is almost impossible to determine how the chromosome structure should be changed in order to encourage feasibility. One simple solution, would be to try new crossover variations from the two parents until a satisfactory candidate is found. For expensive analysis tools such as CFD, this does not provide a satisfactory approach as it could require many samples until a feasible combinant is found. Also, there is a chance that a suitable solution may not even exist between parents.

One solution to this approach, is to use a secondary, computationally cheaper analysis tool to suggest whether a candidate will provide a feasible solution or not. Metamodeling techniques may be used for this task, by learning responses from previous solutions, and using the derived knowledge to evaluate a candidates suitability for CFD analysis based on convergence likelihood. This chapter explores such an approach to assist in the evolution of hi-fidelity geometric representations such as B-Spline curves and orthogonal functions.

8.1 The Effects of Representation on Lethal Crossover

The frequency of finding infeasible solutions when using GA search with CFD, has been well noted by Obayashi [31], Yamamoto and Inoue [30], Jones et al [116], Oyama et al [117], and Periaux et al [33]. The resultant effect of large occurrences of infeasible solutions can disrupt the GA search leading to poor convergence to a weak solution [117]. It was observed during a search of the bump function earlier, that with niching, the number of infeasible solutions that did not contribute to the next generation, increased substantially when population sharing was introduced.

To obtain further insight into the amount of disruption caused by different representation schemes on the quality of the solution space, an investigation into the infeasible solution space has been made for the problems used in Chapter 6. The number of infeasible solutions considered, was measured for an Airfoil Design problem using a GA with a constant population size of 100, and probability of crossover of 60% without niching or clustering. No mutation was used. For the simplest case where B-Spline poles are perturbed from a given design point, the number of infeasible design points required increases substantially with the design space illustrated in Figure 104. Examination of the final drag solution indicates an optimum was successfully located, but without niching, and amongst significant infeasible space, further tests need to be conducted to determine whether the optimum is global or local.

The effect on infeasible points located by different representation schemes was approached in Figure 105. Although the size of the design spaces are different, the use of ortho-normalised functions resulted in the smallest search space, but also in the

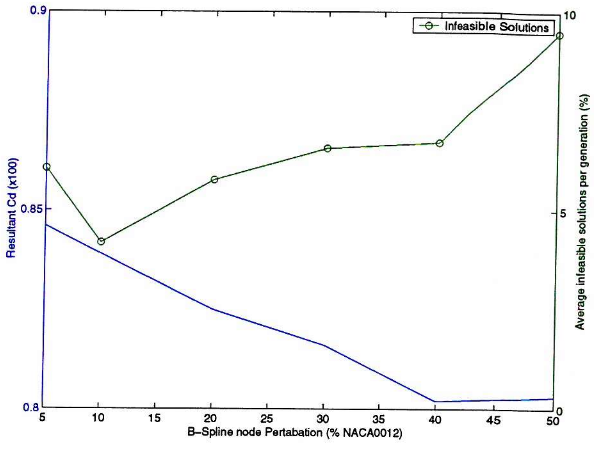

Figure 104: Resultant drag reduction from perturbation method, and the effect of increasing the search space on infeasible solutions

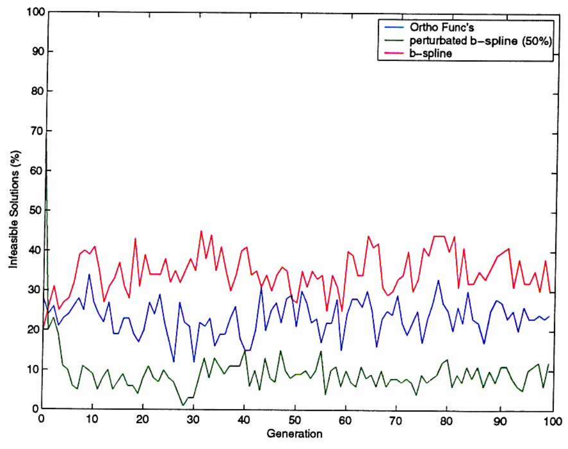

Figure 105: Effect of different representation methods on infeasible solutions

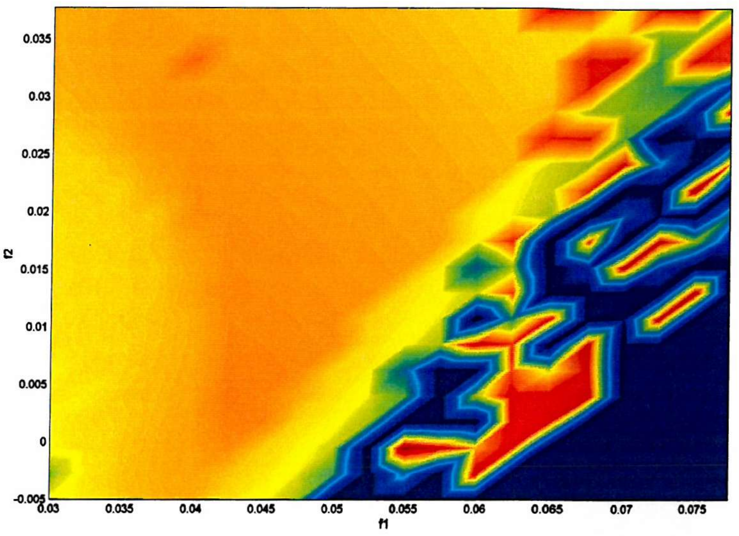

Figure 106: Objective surface for using first two orthogonal modes of the ortho-normalised Aerofunctions at $C_L = 0.6$

highest number of recorded disruptions.

8.2 Neural Network Classification of Infeasible Space

If the fitness landscape were known in advance, expensive CFD computation would no longer be necessary. This is the approach of this section, which considers techniques for modeling the fitness landscape from limited known information. Evolutionary search is ideal for landscape modeling since the population can quickly gather the diverse information needed. To test the type of landscape to be considered, a two dimensional search space is visually shown in Figure 106, based on modifying the first two orthogonal modes of the ortho-normalised aerofunction problem examined in Chapter 6.4.2.

The landscape is effectively divided into three regions, the Infeasible region is coloured black, where a solution could not be obtained using XFoil. The light colour represents low fitness values, which darkens as fitness improves. The optimum is located on an island surrounded by infeasible solutions. Several methods have been used to approximate the fitness landscape obtained through evolutionary search. El-Beltagy [118] showed that a reduction in computational effort can be achieved through this approach. For an initial investigation, the modeling of infeasible space could significantly reduce the computational time of CFD orientated search.

8.2.1 Feed Forward Neural Network

The basic feed forward neural network consists of a layered connection map of non-linear neurons, with the outputs from one layer of neurons, feeding signals into the input of the next layer. By allowing each neuron to express itself as a non-linear sum of weighted input signals, a highly complex non-linear system can easily be created. A typical output from one neuron to another is given as

$$HiddenLayer\ a_j^{(1)} = \sum_l \omega_{jl}^{(1)} x_l + \theta_j^{(1)}; h_j = f^{(1)}(a_j^{(1)}) \tag{107}$$

$$OutputLayer\ a_i^{(2)} = \sum_j \omega_{ij}^{(2)} x_j + \theta_i^{(2)}; y_i = f^{(2)}(a_i^{(2)}) \tag{108}$$

where for example, $f^{(1)}(a) = \tanh(a)$, and $f^{(2)}(a) = a$, and x_j are the neuron input signals from other neurons/inputs, and h_j, y_i are the resultant output signals. Each signal is multiplied by a weight ω_{ij} and a phase shift θ_i.

8.2.2 Classification using a Multi Layer Perception Network

A classification system is easily created from a Multi Layer Perception (MLP) network, by expressing the output neurons as associations to the classification rules. For example, if a MLP model can classify a problem of two classes **a** and **b**, with two output neurons n_1 and n_2, the likelihood of the solution being associated with each class can be inferred by the activation level of the output neurons. Thus, activation levels of $n_1 = 0.7$ and $n_2 = 0.2$ will infer that the input solution is more closely associated with class **a**.

To provide a more definite classification to a given input solution, confidence limits can be placed on the output neuron activation levels. The confidence limit used here, is a threshold limit that an associated output neuron activation must exceed for the input solution to be considered for that class. An input solution will be classified only if at least one output neuron activation level exceeds the threshold limit.

8.2.3 Training the Network

The size of the input variables to a multi layer perception model, must be similar in scale so that the network does not become biased to a dominantly strong input level. To ensure that all input levels are similar, re-scaling of the input parameters is adopted. A simple linear transformation can be used.

$$\tilde{x}_i^n = \frac{x_i^n - \bar{x}_i}{\sigma_i} \tag{109}$$

where $n = 1, ..., N$ labels the training data set, and x_i is a variable for a given training set. $\bar{x}_i$ and σ_i are the mean and standard deviation for that variable with respect to the training set.

In principle, linear scaling for the case of a multi-layer perception model is redundant, since it is often combined with the linear transformation in the first layer of the network. However, it does ensure that all the input and target variables are of the order of unity which provides a suitable range for random initialisation of the weights, prior to network training.

It is important to select a training set that will not cause bias in the multi-layer perception weights. To ensure that an appropriate set is presented for training, a minimal distance criteria is used in that for each one of the training members x^n is at least a distance *dmin* from another member, where the distance between each member is given in Equation 110.

$$distance = \sqrt{\sum_{i=1}^{i=nDim} (\tilde{x}_i^n - \tilde{x}_i^m)^2} \tag{110}$$

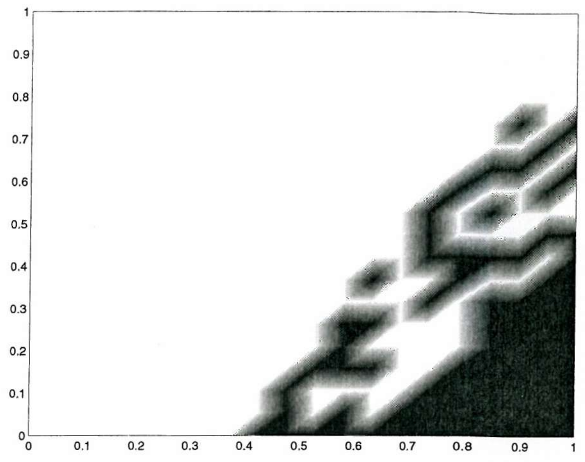

Figure 107: Infeasible Search Space (no solution obtained from Xfoil)

where n and m are the members of the training set between which the distance is being calculated.

8.2.4 Classification of Infeasible Airfoil Shapes

A Multi-Layer Perception model given in [119], has been used to reconstruct the infeasible space shown in Figure 107, by treating it as a classification problem. Using just six hidden neurons, a 93% accurate approximation to the infeasible landscape was achieved in Figure 108 by tightening the decision threshold to 70% output signal strength to define the two answers. Results that did not meet the threshold limits are shown in Gray. The accuracy of the approximation was determined by the number of landscape points that were wrongly classified as either feasible or unfeasible. Points entering the gray area were treated as correct, and would need to be evaluated. Although a reasonable accuracy in results can be maintained through this implementation, a significant reduction of infeasible shape evaluations cannot be expected from using this model.

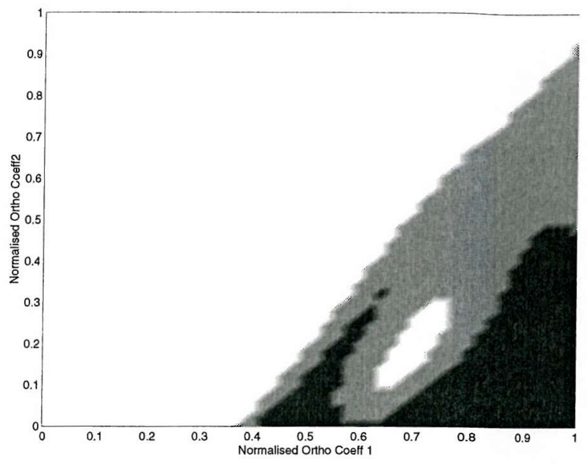

Figure 108: Neural Network Prediction of Infeasible Space

8.3 Meta-Modeling Using a Gaussian Process

Although some measure of accuracy from Forward Feedwork Neural Networks can be used successfully to determine whether a given design point should be evaluated or not, a higher fidelity method is required to acquire the landscape detail lost in Figure 108. El-Beltagy demonstrated the resolution capability of Gaussian processes on such landscapes, and this approach is explored further, for use as a feasible candidate classifier system.

8.3.1 Gaussian Processes

Given a data set D consisting of N pairs of L dimensional input vectors $\mathbf{x}$, we may infer t_{N+1} given the observed scalar output vector $\mathbf{t}_N$. The conditional predicted distribution $P\left(t_{N+1} \mid D, \mathbf{x}_{N+1}\right)$ may be inferred through Equation 111, and can be used to make predictions about t_{N+1}.

$$P\left(t_{N+1} \mid D, \mathbf{x}_{N+1}\right) = \frac{P\left(\mathbf{t}_{N+1} \mid D, \mathbf{x}_{N+1}\right)}{P(\mathbf{t}_N \mid \{\mathbf{x}_N\})} \tag{111}$$

If the joint density $P\left(\mathbf{t}_{N+1} \mid D, \mathbf{x}_{N+1}\right)$ where $\mathbf{t}_{N+1} = \{\mathbf{t}_N, t_{N+1}\}$, is a Gaussian, then the inference of t_{N+1} given $\mathbf{t}_N$ is simple as the conditional distribution

$P\left(t_{N+1} \mid D, \mathbf{x}_{N+1}\right)$, is also a Gaussian.

A Gaussian Process is concerned with evaluating the joint density probability $P\left(t_{N+1} \mid D, \mathbf{x}_{N+1}\right)$ for a given training set D of N pairs of inputs $\mathbf{x}_N$ and scalar outputs t_N. The Gaussian Process is a collection of variables $\mathbf{t} = (t(\mathbf{x}_1), t(\mathbf{x}_2), \cdots)$ which have a joint gaussian distribution given by

$$P\left(\mathbf{t} \mid \mathbf{C}, \{\mathbf{x}_n\}\right) = \frac{1}{\sqrt{(2\pi)^n \left|\mathbf{C}\right|}} \exp\left\{-\frac{1}{2}\left(x-\mu\right)^T \mathbf{C}^{-1}\left(x-\mu\right)\right\} \tag{112}$$

where μ is an n dimensional vector representing the distribution mean, and $\mathbf{C}$ the variance is a $n \times n$ covariance matrix which satisfy,

$$\mu = \varepsilon\left[x\right] \tag{113}$$

$$\mathbf{C} = \varepsilon\left[(x-\mu)(x-\mu)^T\right]. \tag{114}$$

where $\varepsilon[.]$ denotes the expectation. The covariance matrix $\mathbf{C}$ is a parameterised covariance function with hyper-parameters Θ. By solving the hyper-parameters Θ, the conditional distribution $P\left(t_{N+1} \mid D, \mathbf{x}_{N+1}\right)$ can be obtained through relationship 111, and the value t_{N+1} predicted.

This work uses the Gaussian process implemented by Gibbs and MacKay [120], of which a further description of the solution of the hyper-parameters Θ is given in Appendix E.

8.3.2 Gaussian Regression of the Bump Problem

To understand the strengths and weaknesses of using Gaussian Processes in prediction amongst highly modal landscapes, a regression problem is considered here on the fitness surface of the 2D Bump problem given in Chapter 4.1.1. The training set was constructed from N equidistant points across the parameter domain.

A regression problem is considered here to demonstrate the sensitivity of the GP to a highly modal surface. The Bump problem has been reconstructed by GP regression

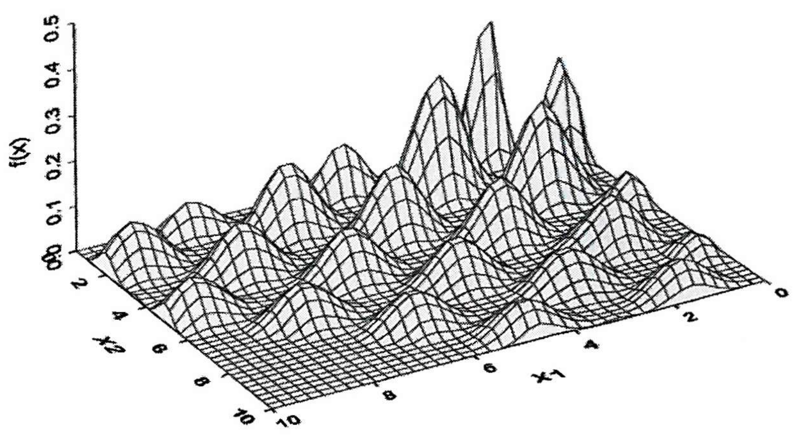

Figure 109: The Bump Function

for 2D, 5D and 20D cases. To measure the accuracy of the surface reconstruction, 1000 random sample points have been compared to the exact solution for each test case.

Figure 109 illustrates the resolution that can be obtained through fitting a bi-cubic spline surface over 400 uniformly sampled points.

By applying a Gaussian Process to the same 400 points, the surface given in Figure 110 is obtained. The accuracy between the surfaces is shown as the in Figure 111, as the difference between the absolute bump surface, and the generated surface using the Gaussian Process.

8.3.3 Integration of Prediction Models with a Genetic Algorithm

The meta-model based on Gaussian Process regression, is to be used in parallel with a GA search, to determine the likelihood of a new member surviving to another generation before expensive computational resources are lost calculating its fitness.

To provide a classification surface through a GP regression model, the problem is

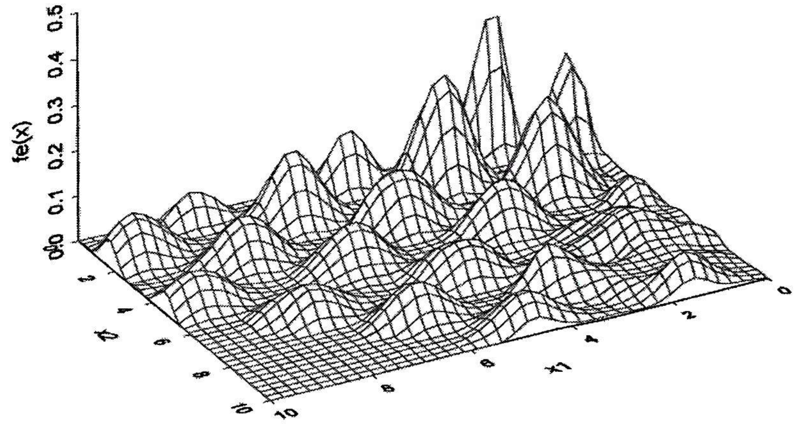

Figure 110: Gaussian Process Metamodel of Bump Function

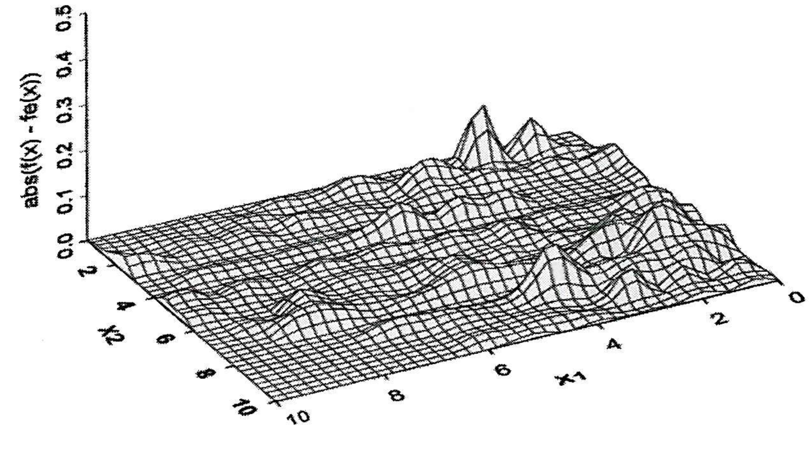

Figure 111: The Absolute Difference Between Bump Metamodel and Bump Function

modeled on the accurate prediction of the fitness surface. For the GP inputs, 20 y co-ordinates are taken from each candidate and presented to the GP pre-processor for normalisation. The y co-ordinates were taken at pre-selected x co-ordinate stations around the airfoil. Co-ordinates were chosen as suitable input parameters instead of the design values being optimised, in order to provide a generic scalable airfoil classification that is independent of problem representation or parameterisation.

Implementation of Genetic Repair Algorithm

The new airfoil classification service was made available to the GA, via JINI-JAVASpace distributed technology used in the distributed GA described in Chapter 7.1.1. In order to maximise genetic sampling productivity with the GA, the classification service is included during the reproduction process. For each time a child fails the classification test, two parents are drawn from the parent pool to reproduce new candidate offspring. This process is continued until sufficient numbers of children pass the initial examination. The candidates are then evaluated as normal by the fitness function. For the distributed classification process, a distributed object is created for each new child. The candidates genetic encoding which implements the curve interface is added to the Distributed Object along with an indicator for the designated analysis model ('gp' or 'xfoil' in this case).

In this work, a pre-trained GP is used for analysis, although a separate training scheme and even on-line training can be incorporated into the JavaSpace implementation, to provide suitable members for the GP, to be included in the training set. A separate process will probably be required to implement this additional feature.

The Implantation algorithm for the integration of distributed classification service is given in Figure 112.

Airfoil Feasibility Classification Service

JINI is a Java dependent networking framework. Components and services can only be made available via JINI as Java Objects. To facilitate the addition of the GP, a Java class was purposely written to act as the Java JINI GP proxy service, passing

```
While $sizeof\ ChildPool < N$
    1. While $sizeof$ Classify Pool $< N$ - $sizeof\ ChildPool$
        (a) Remove 2 Parents from Parent Pool
        (b) Create 2 Children
        (c) Add Children to Classify Pool
    2. Restore Parent Pool
    3. For each in Classify Pool
        (a) Create new Distributed Message Object
        (b) Set Object Message $StatusFlag =$ "Evaluate"
        (c) Add member to Object
        (d) Add Object to JavaSpace
    4. Remove all Distributed Objects from JavaSpace
    5. For Each Distributed Object Retrieved
        (a) If Classification Message = "Feasible" $OR$ "Undecided" $AND\ ChildPool < N$
            • Add Child to $ChildPool$
```

Figure 112: Distributed Genetic Repair and Identification Algorithm

calls to the GP via JNI (Java Native Interface) which enables Java to invoke library methods. In addition to servicing the JavaSpace architecture, the Java wrapper took care of the additional pre GP processing such as calculating the 20 static coordinates from a given section, normalising the GP inputs, and post processing the GP output to determine the airfoil classification. An algorithm describing the Distributed Airfoil Classification service is given in Figure 113

8.4 Application of Airfoil Classification Service to GA Chromosome Repair

For the Airfoil Optimisation problems in Chapter 6, the tasks were re-run using the Airfoil Classification Service to assist in pre-determination of feasible solutions. First, the GP was used as a regression system, offering predictions of the candidates' fitness instead of using CFD analysis. The GP was trained on the actual fitness values scored by 400 members presented, and then used to evaluate each airfoil as the primary and only CFD analysis service to the GA, used instead of XFoil. The convergence results for using both the Bezier Spline representations and ortho-normal curves are shown in Figure 114. In comparison, it is clear that the GP failed to capture the aerodynamic

```
• Initialise GP
• Register for JavaSpace Entry Events where $Evaluator$="Airfoil Classification" $AND$ $StatusFlag$ = "Evaluate";
• Upon Receiving JavaSpace Entry Event for $Evaluator$="AirfoilClassification"
    1. If Contents of Object Received does not support $interface$ Curve
        (a) Reject Object:$StatusFlag$ = "Error"
        (b) ReInsert Distributed Object into JavaSpace
        (c) Dismiss and await next Event
    2. calculate y coordinates at 20 equally spaced parametric curve lengths $0 \leq u \leq 1$
    3. Scale y coordinates as GP input
    4. Get GP Regression for input and rescale output
    5. If output is within failure limit $AND$ variance within variance limit: Classify as Feasible
    6. ElseIf output is not within failure limit $AND$ variance within variance limit: Classify as InFeasible
    7. Else Classify as UnDecided
    8. Add Classification to Distributed Object Message
    9. Set $StatusFlag$ = "Evaluated" on Distributed Object
   10. ReInsert Distributed Object into JavaSpace
```

Figure 113: Distributed Airfoil Classification Service

properties of each section appropriately, leading to convergence on a weak solution.

Similar results were also revealed when an initial training set of 1600 candidates was used. This finding mirrors a similar result found by El-Beltagy who used a GP to construct a meta-model of a Structure design problem. El-Beltagy went on to use online data addition to continuously train the GA Cycle. This additional training data still did not field good GA solutions.

For the Classification Approach, the GA was found to converge faster than that of previous solutions. Both the orthogonal Curve and the Bezier Spline representations, field good GA solutions, with the Bezier surpassing it's original best convergence result. The rapid convergence is most likely due to the more productive and increased rate of schema sampling achieved by using Genetic Screening as a repair service by re-sampling. By screening poor candidate solutions, forcing new chromosome structures to be produced, the schema sample rate is significantly increased. An increase in scheme sampling is more likely to yield good solutions and fast convergence. Another possible affect of candidate screening, is that of increased diversity of good candidates.

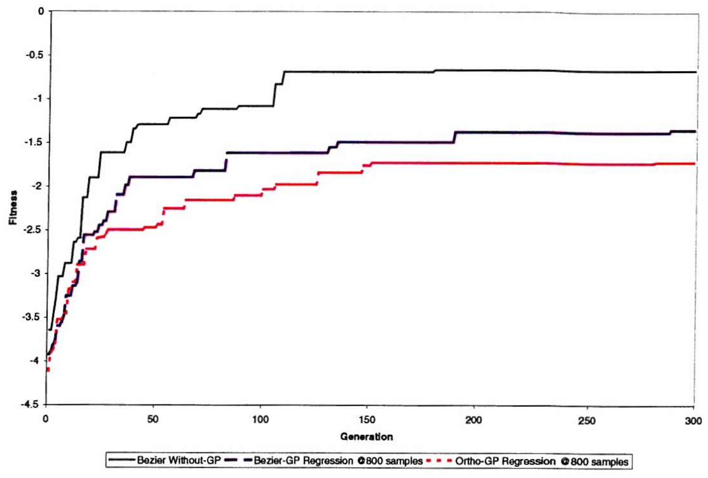

Figure 114: Airfoil Regression

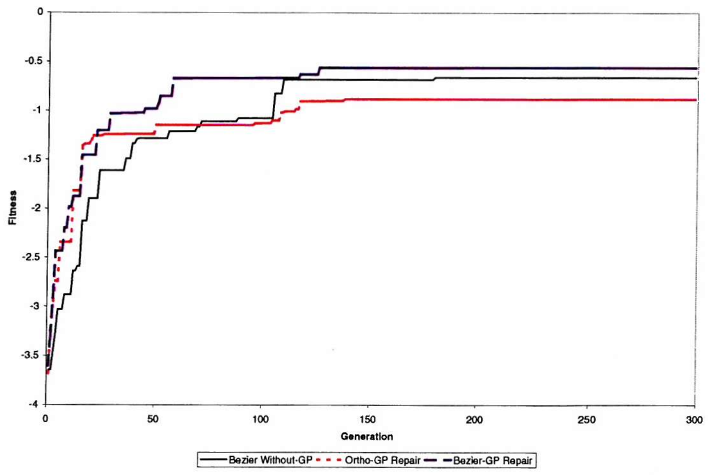

Figure 115: The effect of airfoil repeir

8.5 Summary

Meta modeling Techniques have been considered for assisting a genetic repair service through a genetic screening process during the breeding phase of the GA cycle. Multi-Layer Perception systems were found to be difficult to train as a classification system. This was mainly due to the Bias in the mix of feasible/infeasible candidate solutions available for training. Once a good training set was achieved, good solution accuracy was achieved, although the model was rarely able to completely classify a solution as infeasible, electing for undecided in many cases.

GP is a more accurate model and is less affected by bias in the training set. It can facilitate the use of online data addition to the training set during use, without restarting. It was found to cope well with constructing a regression Meta Model for the Bump problem. However, when used to predict the fitness of the Airfoil sections during a GA cycle, it was unable to capture the fitness surface enough for design/optimisation use.

When applied to an Airfoil Candidate Screening Service, a more healthy convergence was found for the GA. A stronger convergence performance was seen which is most likely attributed to the increase in schema sampling by the GA due to the screening, and more abundant good candidates in the parent pool. To facilitate the Classification Services, the existing JavaSpace was employed in distributing candidates between GA, GP and XFoil, by setting a simple text flag in the distributed object envelope. This allowed the GA to send candidates to the GP using existing infrastructure with minimal modification.

Scalability is always a concern when utilising meta-modeling tools, especially at high dimensions. This work has demonstrated the practical benefit from employing such techniques within an engineering design cycle where the evolution of fitness is an expensive process. However no work has been attempted here to look into scalability issues, which should be examined in more detail before its application into large problems.

Chapter 9

Conclusions and Recommendations

This work has considered the adaptation of aerodynamic shape, and the introduction of complexity adaption within the pseudo chromosome structure used by Genetic Algorithms to evolve design problems. The main conclusions drawn, and important observations, are given below alongside the author's recommendations for further research in the fields discussed.

9.1 Conclusions

The objectives of the work have been met by the following:

1. The rate of successfully crossbreeding species on a difficult objective landscape fell sharply as the maturity of the population increased. On such landscapes, finding the optima often required an element of random fluke, rather than the consequence of evolving numerous small changes. Adapting the behaviour of the environment through adoption of dynamic penalty functions and niching control were found to significantly increase the chance for such random flukes surviving to the next generations and hence finding the optimum.

2. A robust distributed computing platform has been established using networked

dual boot PC clusters with near optimal speedup, achieved using an asynchronous iGA. A new distributed data backup facility was created based on an XOR merge tree. Implementation of this merge tree with a parallel CFD Solver provided an efficient facility for the frequent backup of data across all processes.

3. Spline methods were generally highly tolerant of the number of design points used. Waves in the airfoil pressure distribution frequently observed in the literature where B-Spline representations had been applied, were found to be a result of poor parametric implementation. Orthogonal representations were also investigated and found to be more efficient than that of a simple Bezier curve implementation. Implementation of spline based representations used would benefit from complexity adaptation.

4. A Hierarchical Genetic Algorithm framework for evolving complexity in geometric encodings was established. The chromosome structure was successfully able to adapt its complexity in order to find correct reconstructions of several curves presented, using both segmented lines and B-Spline representations. For the B-Spline case, adaption of knot complexity significantly improved reconstruction performance over traditional fixed gene encodings.

5. When applied to aerodynamic shape optimisation, B-Spline representations evolved quickly to find suitable airfoil shapes that match the order of performance, previously only found using ortho-normalised functions. Both the adaptive ability and evolutionary efficiency of the GA had been greatly enhanced through the adaptation of curve complexity when applied to B-Spline representations.

6. Many airfoil shapes produced by GA analysis could not be examined for aerodynamic performance using the given CFD analysis tool. It is believed that high frequencies of such infeasible solutions degrade the evolution of a GA. A simple repair mechanism was implemented using a Gaussian Process to predict model failure. Using such predictions, new solutions were presented to increase the rate of sampling of the GA. The convergence of the evolution process was found to greatly improve, through the removal of infeasible solutions.

9.2 Further Conclusions and Observations

1. The use of population niching is an essential process that ensures the maintenance of diversity. This diversity may become essential in providing the evolution process new schema hyperplanes, on which to search. Any process that aided the competitive advantage of weak species was found to assist the evolutionary robustness of the search.

2. The evolution process of elite members within a GA, often involves many small leaps in objective function fitness rather than a gradual process of cumulative small changes. Often, elite members created from the result of such leaps, found it difficult to participate in the successful breeding of new members. Any process that reduces the average fitness of the population, thus increasing the probability of new members surviving, would generally improve the evolutionary process.

3. The integration of CFD analysis programs with GAs is difficult as the range of design sample produced by the GA, is generally greater than the evaluation envelope of the analysis tool. A high frequency of unsolvable candidate solutions can disrupt the process of evolution within a GA.

4. Sometimes, the aerodynamic analysis tool used with the GA, will lead to an elite solution that may not be representative of the candidate. Niching will allow several elitist solutions to co-exist and can reduce the problem of misleading solutions propagating. Convergence related penalty functions may also help to remove misleading solutions from the evolution, but at a risk of rejecting good candidates.

5. B-Spline representations are expected to lead to highly evolved shapes. However in practice, the high modality of their implementation was found to produce deceptive search landscapes for the GA to traverse. In contrast, Bezier implementations produced successful results for a wide variety of problems.

6. Ortho-normalised aerofunctions produced though the Gram-Schmidt normalisation of airfoil sections, can lead to a greatly reduced parameter space. Implementation to airfoil design and regression, was slightly prohibited by large fluctuations in numerical noise produced through the discrete implementation of the normalisation process. An improved implementation of ortho-normalisation, of use of improved base geometries may allow the more effective use and analysis of this representation.

7. Complexity adaption produced slower convergence than traditional fixed problem encodings. This can be attributed to the low diversity of genes available for sampling at the beginning of the evolutionary cycle. Crossover efficiency was found to be significantly increased through complexity, particularly in the latter phase of evolution. This is a reversal of the characteristics seen in traditional GA implementations. An application to a previously found deceptive problem, a complexity based GA, was found to evolve more effectively than traditional GAs.

8. Application of the complexity GA to shape optimisation, was found to enhance the robustness of search. Also good solutions were easily obtained using B-Spline based representations. Previous fixed based implementations had required several restarts of the solution to obtain quality results.

9. Regression analysis of shape optimisation landscapes using MLP and GP metamodeling techniques, are unable to capture detail necessary for GA optimisation. Good, accurate classifications of search characteristics were observed, and successfully employed in a simple genetic repair process.

9.3 Recommendations

From the results and observations made throughout this work, the following recommendations are given for further research into some of the subjects covered.

- The problem of species formation and population diversity is key to the success of GA search. The use of resource sharing has offered significant help to the formation of species. The technique used, relied on the ability to identify clusters using an adaptive KMEANS algorithm. Implementation of the KMEANS algorithm requires several key parameters to be defined. The correct definition of niche size, and sharing distance are unknown a priori and are one of the main drawbacks of the KMEANS method. A second problem, is that the accuracy of niche identification by KMEANS, degrades at higher dimensions as demonstrated by Keim [121]. Several improvements to the KMEANS cluster identification algorithm have been proposed, including using an averaged density centred gaussian to identify the cluster, but these all suffered the same fate at high dimensions. A different strategy has been used to cluster data sets in high dimensional space, known as Optigrid [122]. This method is based on mathematical foundations, using the clustering of one dimensional projections of the data set. Using contracting projections, data is merged towards cluster intensities. By finding a point between these intensities allows for simple partitioning of the cluster. This process is repeated within each partition until no more cluster intensities are found.

- Weaker species tend to die out quicker than strong species, which may limit the ability of the GA to traverse search landscape. Interspecies breeding or sexual selection mechanisms that allow members to choose their mates, should be explored in an effort to raise the survival rate of new or weak species.

- The farmer-worker resource sharing scheme, applied through JINI-JavaSpaces, lacked sensitivity to worker failure through the loose asynchronous framework used. Sharing work in this loose fashion has many advantages over traditional direct synchronous procedures, available through RMI and Corba. The better handling of remote network / process / computer exceptions, may allow this approach to eventually become part of a PC background process, accepting processing jobs from a lookup facility similar to JavaSpaces, and processing them when their CPU's are not being used by users. This approach could provide significant processing resources to any office.

- A highly flexible, and powerful distributed computing platform has been provided by coupling office PC's together. Parallel CFD solvers are traditionally implemented on highly efficient parallel libraries. In this work, two robust, commercial standard distributed programming techniques have been implemented (CORBA and Java JINI) to provide a strong, flexible and highly reliable distributed Genetic Algorithm. The performance gain through continuous access to PC computers using such libraries may outweigh that advantage given by highly optimised, but less robust standards such as MPI. Further development would be encouraged in the area of robust parallel computing on NT PCs, that would allow resources to enter or leave parallel CFD process computations, without the need for manual restarts and load-balancing. If automatic load-balancing or partition sharing were available, the use of such PC networks in CFD computation could be harnessed more readily.

- A framework for chromosome structure adaptation has been developed and employed in this work. The variation of encoding complexity was found to assist GA evolution in the field of shape optimisation, and other applications of this idea should be explored.

- The development of complexity in Evolutionary Algorithms is a relatively new field of research [123] and further development in this area, with particular interest to GA application is encouraged.

- Many other ideas such as embryology, borrowed from nature have yet to be applied to evolutionary design. Some limitations have been seen in the costly application of advanced GAs to engineering problems, but with the constant increase in computational power now available to the design office, even some of the most expensive techniques could be useful in this field. Chromosome complexity was found to increase adaptation effectiveness, without increasing GA cost. Preliminary embryology implementations have reported similar observations, although these techniques have yet to be introduced to real engineering problems.

- The evolution of geometric complexity is substantially underdeveloped. Some performance gains given though encoding schemes that adopt this feature have been illustrated by this work. Further experimentation and analysis is strongly encouraged, and may play a pivotal role in the successful application of Genetic Algorithms in the design of complex objects.

REFERENCES

[1] T. Back, D.B. Fogel, and T. Michalewicz. *"Evolutionary Computation 1 basic algorithms and operators"*. Institute of Physics Publishing, Bristol and Philadelphia, 1982.

[2] C.M. Holden and W.A. Wright. "Optimisation methods for wing design". *AIAA, AIAA-00-0842*, 1999.

[3] C. Poloni. "Hybrid ga for multi objective aerodynamic shape optimisation". *Genetic Algorithms in Engineering and Computer Science*, pages 397–415, 1995.

[4] M. Selig and M. Maughmer. "Multipoint inverse airfoil design method based on conformal mapping". *Aiaa*, 30(5):1162–1170, May 1992.

[5] S. Takahashi, S. Obayashi, and K. Nakahashi. "Transonic shock-free wing design with multiobjective genetic algorithms.

[6] R. Dawkins. *"The Blind Watchmaker"*. W.W. Norton and Company, New York London, 1987.

[7] C.R. Darwin. *"On the origin of species"*. John Murray, London, 1959.

[8] C.R. Darwin. *"On the origin of species"*, chapter Chapter 3: 'The Struggle for Existance', pages 71–90. John Murray, London, 1859.

[9] M. Ridley. *"Evolution"*. Blackwell Science, 2nd edition edition, 1996.

[10] B. Lewin. *"Genes VII"*. Oxford University Press, 2000.

[11] D. E. Goldberg. *"Genetic Algorithms in Search, Optimization and Machine Learning"*. Addison-Wesley, 1989.

[12] Z. Michalewicz. *"Genetic Algorithms + Data structures = evolution programs"*. Springer, Berlin, 1996.

[13] J.J. Grefenstette and J.E. Baker. "How genetic algorithms work: A critical look at implicit parallelism". In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA, 1989. Morgan Kaufmann.

[14] G. Syswerda. "Uniform crossover in genetic algorithms". *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9, 1989.

[15] D.E. Goldberg. "Simple genetic algorithms and the minimal deceptive problem". *Genetic algorithms and simulated annealing (Research notes in artificial intelligence)*, 1987.

[16] D.E. Goldberg, B. Korb, and K. Deb. "Messy genetic algorithms: Motivation, analysis, and first results". In *Complex Systems*, volume 3, pages 493–530, 1989.

[17] C. Bishop. *"Neural Networks for Pattern Recognition"*. Oxford University Press, New York, 1995.

[18] J.N. Siddall. *"Optimal Engineering Design:principles and applications"*. Marcel Dekker Inc., New York, 1982.

[19] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *"Numerical recipes in C: the art of scientific computing"*. Cambridge University Press, second edition edition, 1996.

[20] J.A. Nelder and R. Meade. "A simplex method for function minimisation". *Computer Journal*, 7:308–313, 1965.

[21] S. Kirkpatrick, C. Gelatt Jr, and M. Vecchi. "Optimization by simulated annealing". *Science*, 220(4598):671–681, May 1983.

[22] R. Hicks and G. Vanderplaats. "Application of numerical optimization to the design of supercritical airfoils without drag-creep". *Society of automotive engineers*, (770440), March 29-April 1 1977.

[23] A. Jameson. "Aerodynamic design via control theory". *Journal of Scientific Computing*, 3:233–260, 1988.

[24] J. Reuther and A. Jameson. "Aerodynamic shape optimization of wing and wing-body configurations using control theory". In *33rd Aerospace Sciences Meeting and Exhibit*, number AIAA paper 95-0123, Nevada, January 1995. Reno.

[25] A. Jameson, J.J. Alonso, J. Reuther, L. Martinelli, and J.C. Vassberg. "Aerodynamic shape optimization techniques based on control theory". In *29th AIAA Fluid Dynamics Conference*, number AIAA paper 98-2538, Albuquerque, June 1998.

[26] G.W. Burgreen and O. Baysal. "Three-dimensional aerodymanic shape optimization using discrete sensitivity analysis". *AIAA Journal*, 34(9):1761–1969, September 1996.

[27] M.R. Cross. "Aerodynamic design using the euler adjoint approach". In *22nd ICAS Congress*, Harrogate, UK, August 2000.

[28] P. Gage and I. Kroo. "A role for genetic algorithms in a preliminary design environment". *AIAA Paper 93-3933*, 1993.

[29] Y. Crispin. "Aircraft conceptual optimization using simulated evolution". *AIAA Paper 94-0092*, 1994.

[30] K. Yamamoto and O. Inoue. "Applications of genetic algorithm to aerodynamic shape optimization". *AIAA Paper 95-1650*, 1995.

[31] S. Obayashi and T. Tsukahara. "Comparison of optimization algorithms for aerodynamic shape design". *AIAA Paper 96-2394-CP*, pages pp. 181–188., 1996.

[32] S. Aly, M. Ogot, and R. Pelz. "Stochastic approach to optimal aerodynamic shape design". *Journal of aircraft*, 33(5):956–961, sept-oct 1996.

[33] J. Periaux, M. Sefrioui, B. Stoufflet, B. Mantel, and E. Laporte. *Genetic Algorithms in Engineering and Computational Science*, chapter "Robust genetic

algorithms for optimisation problems in aerodynamic design", pages 371–395. J.Wiley, 1995.

[34] D.J. Doorly, J. Peiró, T. Kuan, and J.P. Oesterle. "Optimisation of airfoils using parallel genetic algorithms". *Proceedings of 15th International Conference on Numerical Fluid Mechanics, Monterey*, 1995.

[35] J. Reuther and A. Jameson. "A comparison of design variables for control theory based airfoil optimization". *6th International Symposium on CFD, Lake Tahoe, Nevada*, Sept 1995.

[36] W.A. Wright and C.M.E. Holden. *"Adaptive computing in design and manufacture"*. Spring Verlag London Limited, 1988.

[37] J. Lepine and J. Trepanier. "Wing aerodynamic design using optimised nurbs geometrical regression". *AIAA Paper, AIAA 2000-0669*, 2000.

[38] A.A. Giunta and J. Sobieski. "Progress toward using sensitivity derivatives in a high-fidelity aeroelastic analysis of a supersonic transport". (AIAA paper 98-4763), September 1998.

[39] R.M. Pickett, M.F. Rubinstein, and R.B. Nelson. "Automated structural synthesis using a reduced number of design coordinates". *AIAA Journal*, 11(4):498–494, 1973.

[40] J.P. Leiva and B.C. Watson. "Automatic generation of basis vectors for shape optimization in GENESIS program". *7th AIAA/USAF/NASA/ISSMO symposium on multidiscplinary analysis and optimisation*, (AIAA-98-4852-CP):1115–1122, September 1998.

[41] R.M. Hicks and P.A. Henne. "Wing design by numerical optimization". *Journal of Aircraft*, 15(7), 1978.

[42] J.O. Hager, S. Eyi, and K.D. Lee. "Design efficiency evaluation of transonic airfoil optimization: a case study for navier-stokes design". *AIAA 24th Fluid Dynamics Conference, Orlando*, July 1993.

[43] K. Lee and S. Eyi. "Aerodynamic design via optimization". *Journal of aircraft*, 29(6):1012–1019, nov-dec 1992.

[44] J. Elliott and J. Peraire. "Practical three-dimensional aerodynamic design and optimization using unstructured meshes". *AIAA Journal*, 35(9):1479–1486, 1997.

[45] J. Elliott and J. Peraire. "Constrained, multipoint shape optimisation for complex 3d configurations". *The aeronautical journal*, pages 365–375, august-september 1998.

[46] G. Kiruvila, S. Taásan, and M.D. Salas. "Airfoil design and optimization by the one-shot method". *Proceedings of the 33rd Aerospace Sciences Meeting and Exhibit*, 1995.

[47] S. Cheung. "Parallel optimization of a theoratical minimum-drag body". *6th International Symposium on CFD, Lake Tahoe, Nevada*, pages 193–199, Sept 1995.

[48] M. Drela. "Design and optimization methods for multi-element airfoils". *AIAA/AHS/ASEE Aerospace Design Conference*, February 1993.

[49] J. Reuther, A. Jameson, J.J. Alonso, M.J. Rimlinger, and D. Saunders. "Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers". In *35th Aerospace Sciences Meeting and Exhibit*, number AIAA paper 97-0103, Nevada, 1997. Reno.

[50] G. Farin. *"Curves and surfaces for computer aided geometric design"*. Academic Press, New York, 1990.

[51] V. Braibant and C. Fleury. "Shape optimal design using B-splines". *Computer Methods in Applied Mechanics and Engineering*, pages 247–267, August 1984.

[52] D.J. Doorly, J. Peiró, and J.P. Oesterie. "Optimisation of aerodynamic and coupled aerodynamic structural design using parallel genetic algorithms". *Proceedings of the 6th AIAA USAF ISSMO MDO Conference, AIAA paper 964027CP*, 1996.

[53] M. Ventura and C. Guedes Soares. "Hull form modelling using nurbs curves and surfaces". *Proceeding of the 7th Practical Design of Ships and Mobile Units*, pages 289–295, 1998.

[54] J.A. Samareh. "Use of cad geometry in mdo". *6th AIAA / USAF / NASA / ISSMO symposium on multidiscplinary analysis and optimisation*, (AIAA-96-3991), September 4-6 1996.

[55] J.A. Samareh. "A survey of shape parameterization techniques". *CEAS-AIAA-ICASE-NASA Langley International Forum on Aeroelasticity and structural dynamics*, June 1999.

[56] M.I.G. Bloor and M.J.Wilson. "Generating blend surfaces using partial differential equations". *Computer-Aided Design*, 21(3):165–171, April 1989.

[57] C.W. Dekanski, M.I.G. Bloor, and M.J.Wilson. "The generation of propeller blade geometries using the pde method". *Journal of Ship Research*, 39(2):108–116, June 1995.

[58] M.I.G. Bloor and M.J. Wilson. "Efficient parameterization of generic aircraft geometry". *Journal of Aircraft*, (6):1269–1275, 1995.

[59] P.J. Gage. "New approaches to optimization in aerospace conceptual design". Technical Report NASA Contractor Report 196695, Ames Research Center, Mottett Field, CA, March 1995.

[60] S. Wakayama and I. Kroo. "Subsonic wing planform design using multidisciplinay optimization". *Journal of Aircraft*, 32(4):746–753, july-august 1995.

[61] J.R. Koza, F.H. Bennett, D. Andre, and M.A. Keane. *"Genetic Programming III"*. Morgan Kaufamann, San Francisco, 1999.

[62] P.J. Gage, I.M. Kroo, and I.P. Sobieski. "Variable-complexity genetic algorithm for topological design". *AIAA Journal*, 11(4):498–494, 1973.

[63] P.J. Bentley. *"Generic Evolutionary Design of Solid Objects using a Genetic Algorithm"*. Phd, School of Engineering, University of Huddersfield, 1996.

[64] C. Hirsch. *"Numerical computation of internal and external flows"*, volume 1. John Wiley & Sons, 1990.

[65] N. Hirose, T. Nakamura, and M. Fukuda. "Recent progress on Numerical Wind Tunnel at the national aerospace laboratory, Japan". *Parallel Computational Fluid Dynamics. Recent Developments and Advances Using Parallel Computers*, pages 415–422, 1997.

[66] A.W. Hughes. *"Investigation of tip-driven thruster and waterjet propulsion systems"*. Ph.d. thesis, University of Southampton, 2000.

[67] M. Drela and M.B. Giles. "Viscous-inviscid analysis of transonic and low reynolds number airfoils". *AIAA Journal*, 25(10):1347–1355, 1987.

[68] I.H. Abbott and A.E. Von Doenhoff. *"Theory of wing sections"*. Dover Publications Inc, New York, 1959.

[69] P.L. Roe. "Approximate Riemann solvers, parameter vectors and difference schemes". *Journal of Computational Physics*, 43:357–372, 1981.

[70] N.C. Rycroft. *"An adaptive three-dimensional finite volume Euler solver for distributed architectures using arbitary polyhedral cells"*. Ph.d. thesis, University of Southampton, 1998.

[71] L.J. Eshelman and J.D. Schaffer. "Real-coded genetic algorithms and interval schemata". *Foundations of Genetic Algorithms II*, pages 187–202, 1993.

[72] H. Muhlenbein and D. Schlierkamp-Voosen. "The science of breeding and its application to the breeder genetic algorithm". *Evolutionary Computation*, 1(4):335–360, 1994.

[73] A.J. Keane. "Genetic algorithm optimisation of multi-peak problems:studies in convergence and robustness". *Artificial Intelligence in Engineering*, 9(4):75–83, 1995.

[74] M. Schoenauer and Z. Michalewicz. "Evolutionary computation at the edge of feasibility". *Parallel Problem Solving from Nature, PPSN IV*, 4:245–254, 1996.

[75] Z. Michalewicz, S. Esquivel, R. Gallard, M. Michalewicz, and G. Tao. "Some remarks on design of evolutionary algorithms". *Proceedings of the 3rd On-Line World Conference on Soft Computing in Engineering Design and Manufacture (WSC3)*, 1998.

[76] L. Davis. *"Handbook of Genetic Algorithms"*. Van Nostrand Reinhold, New York, 1990.

[77] D.E. Goldberg. "Genetic algorithms, noise, and the sizing of populations". *Complex Systems*, 6:333–362, 1992.

[78] T. Blickle and T. Lothar. "A comparison of selection schemes used in genetic algorithms". Tik-report, Computer Engineering and Communication Networks Lab, Swiss Federal Institute of Technology, Gloriastrasse 35, 8092 Zurich, Switzerland, 1995.

[79] J. E. Baker. "Reducing bias and inefficiency in the selection algorithm". *Proceedings of 2nd International Conference on Genetic Algorithm*, 1987.

[80] T. Back. "The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm". *Parallel Problem Solving from Nature*, 2:85–94, 1992.

[81] D. Goldberg and K. Deb. "A compararative analysis of selection schemes used in genetic algorithms". *Foundations of Genetic Algorithms*, pages 69–93, 1991.

[82] C.K. Oei, D.E. Goldberg, and Chang S.J. "Tournament selection, niching, and the preservation of diversity". IlliGAL Report No. 91011, Illinois Genetic Algorithms Laboratory, Uinversity of Illinois, Urbana, IL 61801, December 1991.

[83] T. Blickle and T. Lothar. "A mathematical analysis of tournament selection". In L. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 9–16. Morgan Kaufmann, July 1995.

[84] R. Tanese. "Parallel genetic algorithms for a hypercube". In J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, pages 177–183. Lawrence Erlbaum Associates, 1987.

[85] R. Tanese. "Distributed genetic algorithms". *Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, California, USA*, pages 434–439, 1989.

[86] T.C. Belding. "The distributed genetic algorithm revisited". *Proceedings of the Sixth international conference on genetic algorithms*, 1995. Morgan Kaufmann, San Francisco CA.

[87] E. Cantu-Paz. "Topologies, migration rates, and multi-population parallel genetic algorithms". IlliGAL Report No. 99007, Illinois Genetic Algorithms Laboratory, Uinversity of Illinois, Urbana, IL 61801, January 1997.

[88] D.E. Goldberg and J. Richardson. "Genetic algorithms with sharing for multimodal function optimization". *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, 1987.

[89] S.W. Mahfoud. "Niching methods for genetic algorithms". IlliGAL Report No. 99007, Illinois Genetic Algorithms Laboratory, Uinversity of Illinois, Urbana, IL 61801, May 1995.

[90] K. Deb and D.E. Goldberg. "An investigation of Niche and Species formation in genetic function optimization". *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, 1989.

[91] M. Pelikan and D. Goldberg. "Genetic algorithms, clustering, and the breaking of symmetry". IlliGAL Report No. 2000013, Illinois Genetic Algorithms Laboratory, Uinversity of Illinois, Urbana, IL 61801, March 2000.

[92] J.E. Beasley and P.C. Chu. "A genetic algorithm for the set covering problem". *European Journal of Operational Research*, 94:392–404, 1996.

[93] X. Yin and N. Germay. "A fast genetic algorithm with sharing scheme using cluster methods in multimodal function optimization". *Proceedings of the International Conference on artificial neural nets and genetic algorithms*, pages 450–457, 1993.

[94] K.A. DeJong and W.M. Spears. "An analysis of the interacting roles of population size and crossover in genetic algorithms". *Parallel Problem Solving from Nature, In H.P.Schwefel and R.M´anner*, pages 38–47, 1990.

[95] S.R. Turnock. *"Prediction of ship rudder-propeller interaction using parallel computations and wind tunnel tests"*. Ph.d. thesis, University of Southampton, 1993.

[96] W. Gropp, E. Lusk, and N. Doss. "A high performance, portable implementation of the mpi message passing interface standard". *Parallel Computing*, 22(6):789–828, 1996.

[97] D.A. Nicole, K. Takeda, and I.C. Wolton. "HPC on DEC Alpha and Windows NT". *Proc. HPCI Conf. 98*, Manchester 12-14 1998.

[98] K. Takeda, O.R. Tutty, and D.A. Nicole. "Parallel discrete vortex methods on commodity supercomputers; An investigation into bluff body far wake behavior". *Proc. 3rd International Workshop on Vortex Flow and Related Numerical Methods*, Toulose, August 1998.

[99] N.C. Rycroft and S.R. Turnock. "Hybrid cell finite volume Euler solutions of flow around a Main-Jib sail using an IBM SP2". *Parallel Computational Fluid Dynamics. Recent Developments and Advances Using Parallel Computers*, pages 263–272, 1997.

[100] W. Gropp, E. Lusk, and A. Skjellum. *"Using MPI. portable parallel programming with the Message Passing Interface"*. MIT Press, 1994.

[101] B.L. Bihari, V. Shankar, and S. Palaniswamy. "Massively parallel implementation of an explicit CFD algorithm on unstructured grids, II". *Parallel Computational Fluid Dynamics. Recent Developments and Advances Using Parallel Computers*, pages 241–248, 1997.

[102] Message Passing Interface Forum. "MPI: A message-passing interface standard". Technical Report CS-94-230, University of Tennessee, Knoxville, TN, 1994.

[103] C. Walshaw, M. Cross, and M. Everett. "Mesh partitioning and load-balancing for distributed memory parallel systems". *Proc. Parallel and Distributed Computing for Computational Mechanics*, 1997.

[104] T.H. Abbott and A. Doenhoff. *"Theory of Wing Sections"*. Dover Publications, 1949.

[105] M.J.D. Powel. "An efficient method for finding the minimum of a function of several variables without calculating derivatives". *Computer Journal*, 7(4):304–307, 1964.

[106] I. Chang, F.J. Torres, and C Tung. "Geometric analysis of wing sections". NASA Technical Memorandum 110346, Ames Research Center, NASA, Moffett Field, California, April 1995.

[107] M.W.C. Oosterveld. "Ducted propeller characteristics". *Symposium on Ducted Propellers*, pages 35–69, 1973.

[108] M. Selig. "UIUC airfoil data site". http://amber.aae.uiuc.edu/ m-selig/ads.html, 2002.

[109] K.F. Man, K.S. Tang, and S. Kwong. *"Genetic Algorithms"*. Springer, 1999.

[110] P.J. Bentley and J.P. Wakefield. "Hierarchical crossover in genetic algorithms". *Proceedings of the 1st online workshop on soft computing (WSC1)*, 1996.

[111] J. Gosling, B. Joy, G. Steele, and G. Bracha. *"The Java Language Specification, Second Edition"*. Addison Wesley, 2000.

[112] J. Siegel. *"Corba 3: Fundamentals and Programming"*. John Wiley and Sons, 2000.

[113] D. Gelernter. "Generative communication in linda". *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.

[114] E. Cohen, T. Lyche, and R.F. Riesenfield. "Discrete B-splines and subdivision techniques in Computer Aided Design and computer graphics". *Computer Graphics and Image Processing*, 14:87–111, 1980.

[115] W. Boehm. "Inserting new knots into B-spline curves". *Computer Aided Design*, 12:199–201, 1980.

[116] B.R. Jones, W.A. Crossley, and A.S. Lyrintzis. "Aerodynamic and aeroacoustic optimization of airfoils via a parallel genetic algorithm". *AIAA Paper, AIAA 98-4811*, 1998.

[117] S. Oyama, S. Obayashi, K. Nakahashi, and T. Nakamura. "Transonic wing optimization using genetic algorithm". *AIAA Paper 97-1854*, 1997.

[118] M.A. El-Beltagy. *"A natural approach to multilevel optimisation in engineering design"*. Ph.d. thesis, University of Southampton, 1999.

[119] T.Masters. *"Practical Neural Network Recipes in C++"*. Academic Press, 1993.

[120] M.N. Gibbs. *"Bayesian Gaussian Processes for Regression and Classification"*. Phd, University of Cambridge, 1997.

[121] A. Hinneburg and D.A. Keim. "An efficient approach to clustering in large multimedia databases with noise". *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, AAAI Press*, 1998.

[122] D.A. Keim and A. Hinneburg. "Optimal grid-clustering towards breaking the curse of dimensionality in high-dimensional clustering". *Proceedings of the 25th VLDB Conference*, 1999.

[123] A.K. Seth. "The evolution of complexity and the value of variability". In C. Adami, R.K Belew, H. Kitano, and C.E. Taylor, editors, *Artificial Life VI: Proceedings of the Sixth International Conference on the Simulation and Synthesis of Living Systems*, pages 209–221, Cambridge, MA, 1998. MIT Press.

APPENDICES

Appendix A

Adaptive KMEANS Algorithm

Adaptive MacQueen's KMEANS algorithm allows the number of clusters to vary during initial assignment of data members. To control cluster formation, two parameters d_{min}, the *minimal* distance between cluster centres, and d_{max}, the *maximal* radius of the cluster, are required before cluster analysis.

To define the size and position of clusters, the following proceedure is used.

1. For an initial number of clusters k, take the first k data units and assign each one to a seperate cluster. Calculate all pairwise distances between cluster centroids, and merge clusters together if $pairwise\ distance < d_{min}$. Continue merging clusters while this condition is satified.

2. Assign remaining data points to nearest clusters. If $member\ pairwise\ distance > d_{max}$ for all cluster - point relationships, assign point to a new cluster. Attempt to merge clusters after each point insertion.

3. Take all centroids as fixed seed points, and reassign all data members to nearest centroid.

To enchance the use of this algorithm with a GA, the points are first sorted in relation to their fitness.

Appendix B

Genetic Algorithm Implementation

The Genetic Algorithm described in Chapter 4, has been implemented using the object relationship described loosely in Figures 116 to 117. These diagrams only show hierarchical relationships between key classes in order to simplify this description.

The Population class manages each population of members throughout the evolution cycle. Individuals are represented through the Genome class, which contains references to their encoding, and objective function. Thus a Genome is able to evaluate itself, as well as to pass calls to its recombination processes (Crossover, Inversion and Mutation) on to their implementing Representation class. The key breeding processes are controlled by a separate Breeding class, and the Genetic Algorithm acts as a administrative object for controlling and documenting the complete evolution of a population. For the island implementation, involving several GAs, an Immigration-Control object referenced by all Generic Algorithm instances is used as an intermediate to pass member Genome's from one GA to another.

B.1 Genome Management Classes

GeneticAlgorithm: The GA is responsible for initialising and managing the evolution process. From a template Chromosome structure, the GA creates a number of Genomes either by perturbating the members design values, or from file. The members are loaded into a Population container class where the members initial

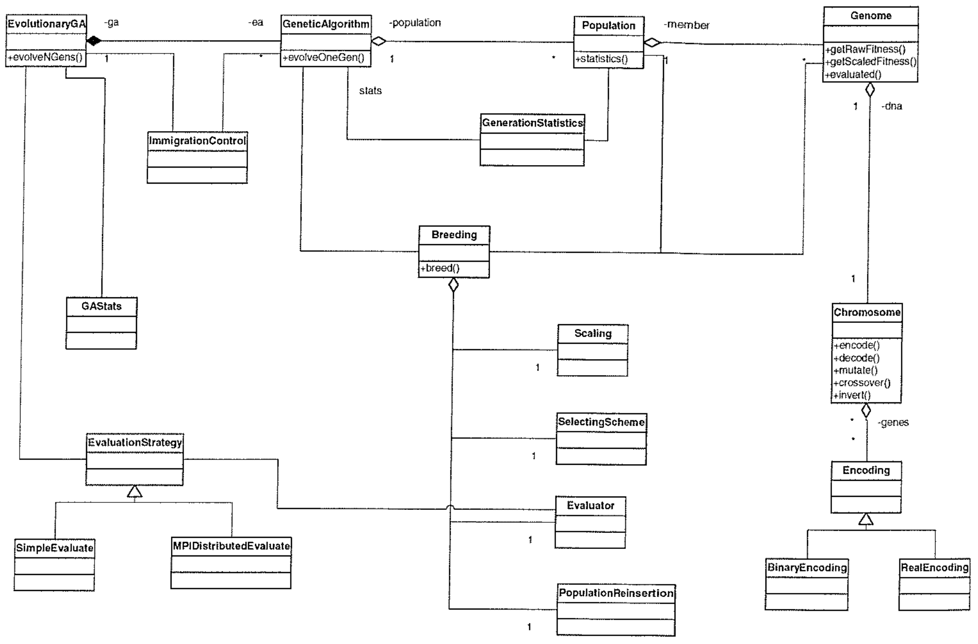

Figure 116: A Simplified UML diagram showing hierarchical relationships of the main classes that implement the GA

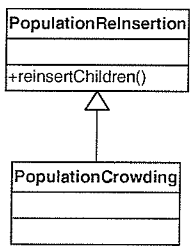

Figure 117: Population reinsertion scheme implementation for Crowding control

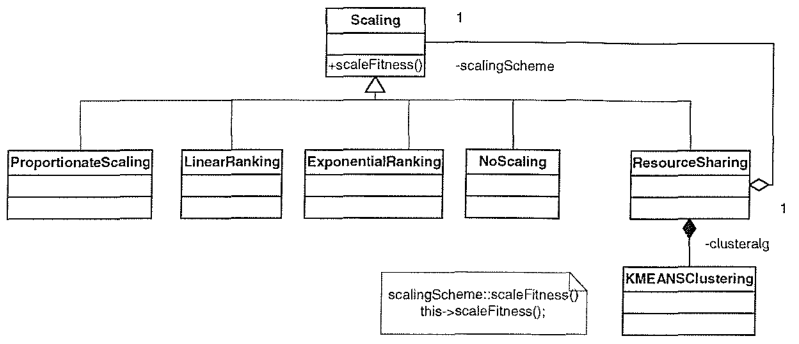

Figure 118: Genome fitness scaling implementation UML

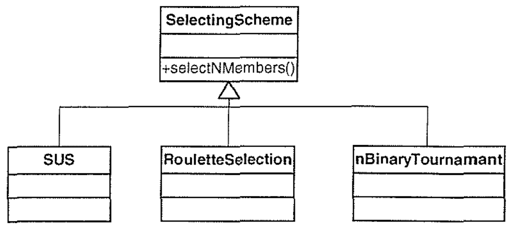

Figure 119: Parent selection scheme implementation UML

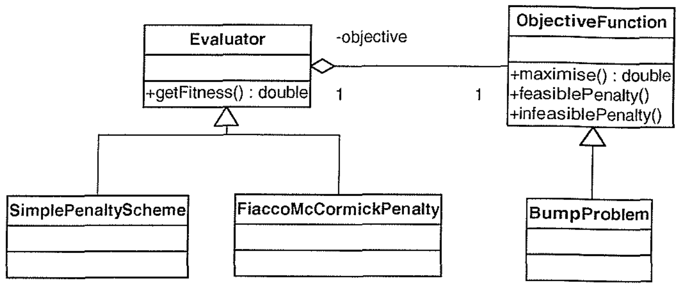

Figure 120: UML diagram of fitness evaluation via a penalty scheme

fitness is evaluated. The Population is evolved generation by generation until the maximum number of generations is reached. At each generation, population statistics are measured and recorded.

Population: The Population is a container class of Genome objects. It is mainly responsible for managing access to individual members.

Genome: The Genome class represents an individual in the Population, and contains important evolutionary information such as its fitness and dna encoding. The Genome is able to breed with other Genomes through crossover, mutation and inversion operators that pass calls on to its dna encoding.

GenerationStatistics: The vital statistics of each generation are recorded in this object. In this implementation, Population size, best fitness, average fitness, fitness variance, generation, elite member index and elitist members details are recoded.

EvolutionaryGA: In order that several instances of GA can evolve their populations in parallel, EvolutionaryGA represents the top most level Object called from the main function. This object manages the iGA implementation.

GAStats: This class is required to record generational statistics for multiple GA instances.

ImmigrationControl: This class manages the migration process of member Genomes from one GA instance to another in an iGA implementation. GA's pass members for migration to the ImmigrationControl instance, where it is stored until retrieved by its destined GA instance.

EvaluationStrategy: This proxy provides a means for distributed processing of Genome fitness evaluation. For the MPIDistributedEvaluate class, a farmer-worker farm distribution strategy is used with asynchronous coupling.

B.2 Breeding Classes

The relationship of classes associated with specialised breeding control are loosely described in Figures 118 and 117.

Scaling: This class scales a Genome's fitness by some amount depending on its performance relative to the other members of the population. The type of scaling applied is defined on the generalisation of this class, but typically rank selection is used as a default implementation.

ResourceSharing: Niching control is effectively applied through resource sharing, which reduces the scaled fitness of an individual depending on the phenotypical distance between to the niche centre and the number of members sharing the niche.

KMEANSClustering: KMEANS clustering implementation used to group Genomes into relative resource niches.

SelectingScheme: This class is responsible for parental selection. Three schemes are implemented; SUS, Roulette and Tournament Selection.

Evaluator: The Evaluator class calculates the Genome's fitness using the appropriate Penalty scheme. The raw fitness assigned to the Genome is usually some function of objective function fitness minus some penalty reflecting the non-feasibleness of the Genome.

ObjectiveFunction: The objective function is an abstract class intended to be generalised by a user's implementation of their objective problem. In this implementation, on maximisation is considered

PopulationReInsertion: Once the children have been created and evaluated, they are normally re-inserted back into the final population via this class. If elitist strategies are to be employed, they would normally be defined here.

PopulationCrowding: A population reinsertion implementation where children attempt to replace the phenotypically closest parents found in small random sampling of some predefined size.

B.3 Genetic Encoding Classes

Chromosome: The Chromosome represents the Genotype DNA encoding that is vital for the Genetic Algorithm evolution process. The main function of the Chromosome class is the manipulation of the genes components via crossover, mutation and inversion. The Chromosome is also able to encode and decode the gene strings.

Encoding: The encoding represents the Genotype-Phenotype map used to encode and decode the genes contained in the chromosome, into real design values. In this implementation, each gene has one encoding instance associated.

BinaryDNA: A binary string generalisation of an encoding, that uses grey coding to reduce the effect of hamming distance.

RealEncoding: A real number encoding generalisation offering an alternative to binary strings.

Appendix C

Normal Mode Analysis

A Gram-Schmidt procedure for ortho-normalisation can be developed from the property of orthonormal functions such that,

$$\begin{gathered} \int_0^1 f_m(x) f_n(x) dx = 0 \quad (m \neq n) \\ \int_0^1 f_m^2(x) dx = 1 \end{gathered} \tag{115}$$

Let $g_k(x)$ be the functions that are not orthogonal. Then the orthogonal set $\bar{f}_k(x)$ is formed from the following relations:

$$\begin{aligned} \bar{f}_1(x) &= g_1(x) \\ \bar{f}_2(x) &= a_{21}\bar{f}_1(x) \\ &\cdot \\ &\cdot \\ \bar{f}_k(x) &= \sum_{m=1}^{k-1} a_{km}\bar{f}_m(x) \\ &\cdot \\ &\cdot \end{aligned} \tag{116}$$

where a_{km} is the projection of g_k in the direction of $\bar{f}_m$,

$$a_{km} = -\frac{\int_0^1 g_k(x)\bar{f}_m(x)dx}{\int_0^1 \bar{f}_m^2(x)dx}. \tag{117}$$

Finally, the orthonormal functions are found by normalising $\bar{f}_k(x)$ as follows:

$$f_k = \frac{\bar{f}_k(x)}{\sqrt{\int_0^1 \bar{f}_k^2(x)dx}}. \tag{118}$$

Appendix D

Java Implementation of Adaptive Chromosome Encoding

The Java implementation of the Genetic Algorithm, uses a class library similar to that detailed in Appendix B, which was made possible through the relatively straight forward conversion between the C++ code, and Java implementation. One important modification introduced, is a Java Interface between the Genome, and its Chromosome encoding. The interface used, 'DNAEncoding', separates Chromosome specific implementation from the genetic operations and other methods required of an encoding by the Genetic Algorithm. Using this interface, the Genetic Algorithm is able to pass calls to the encoding, regardless of its implementation.

A simplified UML diagram is shown in Figure 121, to illustrate the relationship between the chromosome classes and the 'DNAEncoding' interface.

In Figure 121, GenoCurve is an implementable chromosome encoding and thus implements the DNAEncoding interface. The lower level component of GenoCurve, is the GenoPoint class which represents a mutable implementation of a three dimensional point. Although these classes belong to the chromosome encoding family (Genotype), they also contain characteristics of geometry (phenotype). Further phenotype implementations can be built from these base classes. The complexity based B-Spline chromosome implementation is shown in Figure 122, and the airfoil implementation based on the complex B-Spline representation is illustrated in Figure 123

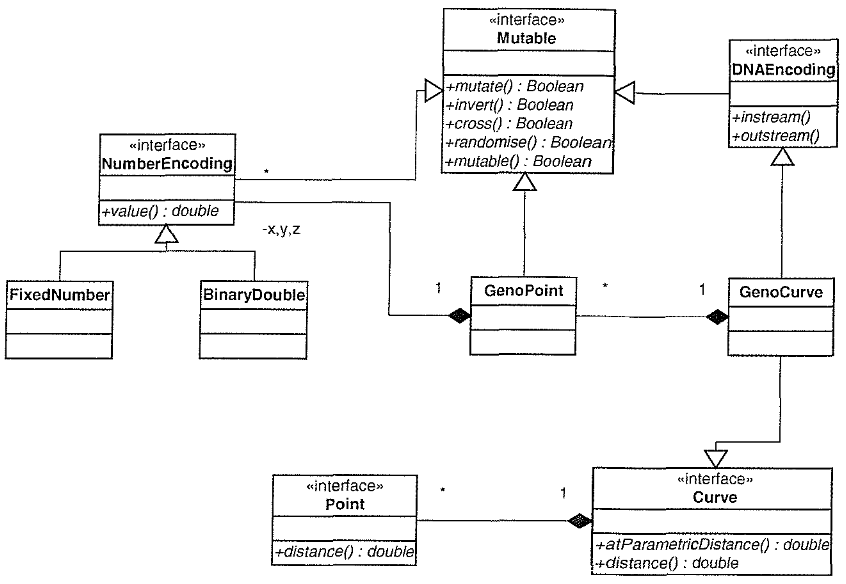

Figure 121: UML Diagram of the Hierarchical Chromosome Encoding Framework Used to Adapt Encoding Complexity

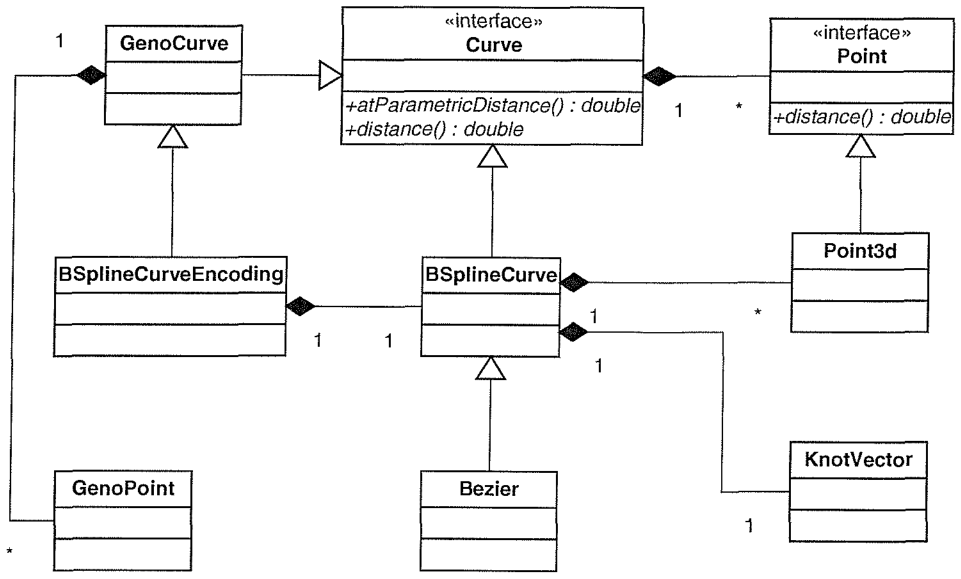

Figure 122: UML Diagram of the Complexity Based B-Spline Encoding Implementation

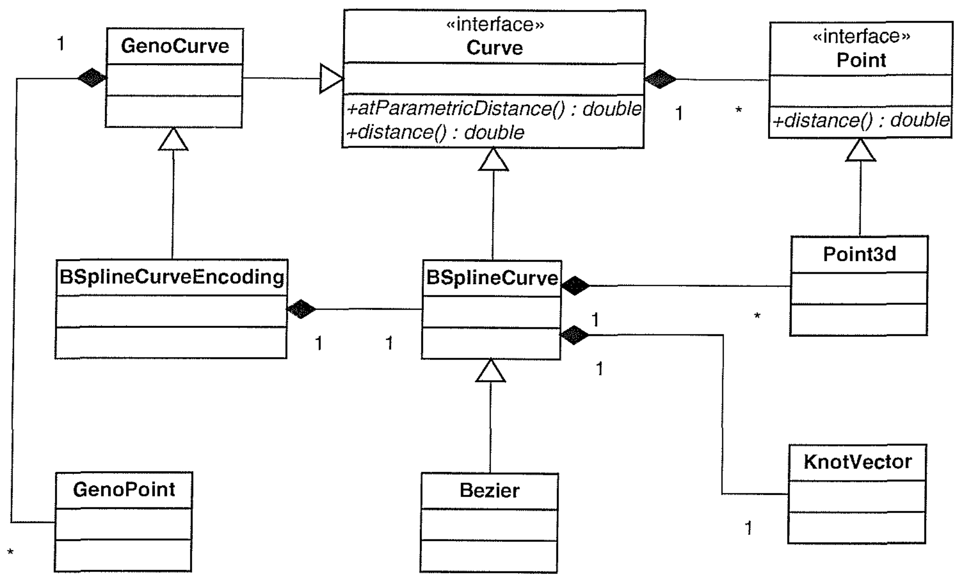

Figure 123: Airfoil Encoding Using Complexity Based B-Splines for the Upper and Lower Curves

Appendix E

Implementation of Gaussian Process

For the conditional Gaussian distribution given by

$$P\left(\mathbf{t} \mid \mathbf{C}, \{\mathbf{x}_n\}\right) = \frac{1}{\sqrt{(2\pi)^n \left|\mathbf{C}\right|}} \exp\left\{-\frac{1}{2}\left(\mathbf{t}-\mu\right)^T \mathbf{C}^{-1}\left(\mathbf{x}-\mu\right)\right\}, \tag{119}$$

the predictive probability distribution for t_{N+1} is

$$\begin{aligned} P\left(t_{N+1} \mid D, \mathbf{x}_{N+1}\right) &= \tfrac{P(\mathbf{t}_{N+1} \mid D, \mathbf{x}_{N+1})}{P(\mathbf{t}_N \mid \{\mathbf{x}_{N+1}\})} \\ &= \tfrac{1}{\sqrt{(2\pi)\frac{|\mathbf{C}_{N+1}|}{\mathbf{C}_N}}} exp\left[-\tfrac{1}{2}\left(\mathbf{t}_{N+1}^T \mathbf{C}_{N+1}^{-1} \mathbf{t}_{N+1} - \mathbf{t}_N^T \mathbf{C}_N^{-1} \mathbf{t}_N\right)\right]. \end{aligned} \tag{120}$$

The covariance matrix for $\mathbf{C}_{N+1}$ is $\mathbf{C}_N$ plus an additional column and row:

$$\mathbf{C}_{N+1} = \left[\begin{array}{c|c} \mathbf{C}_N & \mathbf{k} \\ \hline \mathbf{K}^T & k \end{array}\right] \tag{121}$$

where the $\mathbf{K}$ vector and k scalar are defined as

$$\mathbf{K}^T = \left[C(\mathbf{x}_1, \mathbf{x}_{N+1}), \cdots, C(\mathbf{x}_N, \mathbf{x}_{N+1})\right], \tag{122}$$

$$k = C(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}). \tag{123}$$

By collecting the terms that are a function of t_{N+1} in Equation 120, the Gaussian distribution can be expressed as

$$P\left(t_{N+1} \mid D, \mathbf{x}_{N+1}\right) = \frac{1}{\sqrt{(2\pi)\frac{|\mathbf{C}_{N+1}|}{\mathbf{C}_N}}} exp\left[-\frac{\left(t_{N+1} - \hat{t}_{N+1}\right)^2}{2\sigma^2_{\hat{t}_{N+1}}}\right], \tag{124}$$

where

$$\hat{t}_{N+1} = \mathbf{k}^T\mathbf{C}_N^{-1}\mathbf{t}_N, \tag{125}$$

$$\sigma^2_{\hat{t}_{N+1}} = k - \mathbf{k}^T\mathbf{C}_N^{-1}\mathbf{t}_N. \tag{126}$$

Hence, the prediction given by $\hat{t}_{N+1}$, $\sigma^2_{\hat{t}_{N+1}}$ provide a measure of the confidence in the prediction.

Elements of the covariance matrix $\mathbf{C}_N$ are calculated using the covariance function $\mathbf{C}(\mathbf{x_i}, \mathbf{x_j})$. Thus $(\mathbf{C_N})_{\mathrm{ij}} = \mathbf{C}(\mathbf{x_i}, \mathbf{x_j})$. The covariance function used, is given by

$$C\left(\mathbf{x}_i, \mathbf{x}_j\right) = \theta_1 exp\left[-\frac{1}{2}\sum_{l=1}^{L}\frac{\left(x_i^{(l)} - x_j^{(l)}\right)}{r_l^2}\right] + \theta_2 + \delta_{ij}\theta_3 \tag{127}$$

where x_n^l is the l_{th} component of x_n and the hyperparameters are defined as $\Theta = log(\theta_1, \theta_2, \theta_3, \mathbf{r})$ and are used to control the scale, correction bias, noise level, and input difference. The hyperparameters are defined as the log of the variables in Equation 127 to restrict their values to be positive.

Using Bayes theorem, the posterior probability of the hyperparameters given the training data is

$$P(\theta \mid D) = \frac{P\left(\mathbf{t}_N \mid \{\mathbf{x}_N\}, \theta\right) P\left(\theta \mid \{\mathbf{x}_N\}\right)}{P\left(\mathbf{t}_N \mid \{\mathbf{x}_N\}\right)}. \tag{128}$$

Rather than maximising Equation 128 directly to determine the maximum *a posteriori* estimate for Θ, the logarithm of the probability is maximised. The logarithm of the posterior probability is hence

$$L = -\frac{1}{2} log\,|\mathbf{C}_N| - \frac{1}{2}\mathbf{t}_N^T \mathbf{C}_N^{-1} \mathbf{t}_N - \frac{N}{2} log 2\pi. \tag{129}$$

The maximisation is usually carried out using a gradient based optimiser where the gradient is given by

$$\frac{\partial L}{\partial \theta} = -\frac{1}{2} trace \left(\mathbf{C}_N^{-1} \frac{\partial \mathbf{C}_N}{\partial \theta} \right) + \frac{1}{2}\mathbf{t}_N^T \mathbf{C}_N^{-1} \frac{\partial \mathbf{C}_N}{\partial \theta}. \tag{130}$$

Once the hyperparameters are obtained, the covariance matrix can be used to determine the prediction for t_{N+1}, and the confidence factors given in Equations 125 and 126.